Theses and Dissertations   1. Thesis and Dissertation Collection, all items

1994-09

Integrating middleware software into open-system client/server systems

Weiss, Karen L.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/35215

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

INTEGRATING MIDDLEWARE SOFTWARE INTO OPEN-SYSTEM CLIENT/SERVER SYSTEMS

by

Karen L. Weiss

September 1995

Thesis Advisor: Barry Frew

19960215 016

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 3

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1995	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE INTEGRATING MIDDLEWARE SOFTWARE INTO OPEN-SYSTEM CLIENT/SERVER SYSTEMS	5. FUNDING NUMBERS
6. AUTHOR(S) Weiss, Karen L.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The Corporate Information Management (CIM) initiative formalizes the goals for information systems management within DOD. Four of the six goals CIM lists call for systems integration and data sharing. These goals will be difficult to accomplish because each service has unique requirements and multiple information systems supporting similar tasks (Clancy, 1994). The growing need for integration solutions has spawned a new software category called middleware. The capabilities of middleware hold great promise for increasing system functionality and rejuvenating legacy systems while decreasing development time. "The future belongs to those with the tools and processes to turn mission-critical distributed data into knowledge and competitive assets (Barbagallo, 1994)."

This research presents an analysis of and explores the current state of middleware software development tools. This research also presents the lessons learned from an application development project using a middleware tool.

14. SUBJECT TERMS Client/Server, Middleware, Distributed Processing, Delphi	15. NUMBER OF PAGES 106
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

INTEGRATING MIDDLEWARE SOFTWARE
INTO OPEN-SYSTEM CLIENT/SERVER SYSTEMS

Karen L. Weiss
Lieutenant, United States Navy
B.S., University of the State of New York, 1990

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 1995

Author: _____
Karen L. Weiss

Approved by: _____
Barry Frew, Thesis Advisor

_____
Suresh Sridhar, Associate Advisor

_____
Reuben T. Harris, Chairman
Department of Systems Management

iii

ABSTRACT

The Corporate Information Management (CIM) initiative formalizes the goals for information systems management within DOD. Four of the six goals CIM lists call for systems integration and data sharing. These goals will be difficult to accomplish because each service has unique requirements and multiple information systems supporting similar tasks (Clancy, 1994). The growing need for integration solutions has spawned a new software category called middleware. The capabilities of middleware hold great promise for increasing system functionality and rejuvenating legacy systems while decreasing development time. "The future belongs to those with the tools and processes to turn mission-critical distributed data into knowledge and competitive assets (Barbagallo, 1994)."

This research presents an analysis of and explores the current state of middleware software development tools. This research also presents the lessons learned from an application development project using a middleware tool.

TABLE OF CONTENTS

I. INTRODUCTION

Increasingly, organizations are encountering situations in which heterogeneous, autonomous, distributed computing systems need to be connected and must interoperate to provide services and information. For example, re-engineering of business processes might require the integration of existing (legacy) business systems with new applications; the use of ubiquitous networks, such as the Internet, to connect heterogeneous computing sites; and a mobile system to communicate with different computing sites as it moves about the globe. For many of Department of Defense's (DOD) information managers, these integration scenarios would require the invention of creative, custom connectivity solutions. But integration problems are not unique to the DOD. The growing need for integration solutions has spawned a new software category called middleware.

The capabilities of middleware hold great promise for increasing system function-ality and rejuvenating legacy systems while decreasing development time. "The future belongs to those with the tools and processes to turn mission-critical distributed data into knowledge and competitive assets (Barbagallo, 1994)." DOD has recognized the need to update its legacy systems and reduce the number of stovepipe systems. The Corporate Information Management (CIM) initiative formalizes the goals for information systems management within DOD. CIM lays out six basic goals: (Endoso, 1994)

- Re-invent and re-engineer DOD's processes,
- Couple DOD organizations together through common, shared data ,
- Minimize duplication and enhance DOD's information systems,
- Implement a flexible, world-wide information infrastructure,
- Apply CIM to integrate DOD-wide operations,
- Establish a CIM policy and management structure.

Four of the six goals listed above call for systems integration and data sharing. These goals will be difficult to accomplish because each service has unique requirements and multiple information systems supporting similar tasks (Clancy, 1994). At one point there were more than 30 different civilian payroll systems in the DOD (Clancy, 1994). User requirements are frequently unclear and change as the user becomes familiar with the

1

system. Each system requires different applications and different connections (Booch, 1994). These enterprise-wide systems usually require some custom software development to interconnect commercial off the shelf (COTS) applications together (Booch, 1994). These system-unique elements make enterprise-wide systems difficult to develop.

However, when agencies come to depend on the benefits and resources that integrated systems can deliver - interorganization coordination and vulnerability are increased (Hart, P. and Estrin, D., 1991). The mere presence of the system will change the environment (Booch, 1994) and their very use shifts the nature of interdependence (Hart, P. and Estrin, D., 1991). This phenomenon emphasizes the need to use development tools that are flexible, easy to use, and enable rapid application development (RAD). While there is no magic map for achieving an open system client/server environment, the critical piece is middleware (Orfali & Harkey, 1994).

A. PURPOSE OF THESIS

The purpose of this research is to present an analysis of and explore the current state of middleware software development tools. Middleware software was developed to solve interoperability problems and provide users with the data manipulation flexibility they demand. Regardless of commercial vendor claims, no product, or group of products, is a silver bullet which will solve all system integration problems. Therefore, an understanding of this new technology is essential if DOD is to exploit its capabilities while respecting its limitations. This research will address the following questions:

- What distributed processing considerations are important when redesigning enterprise systems to include the use of middleware software?
- What types of applications can middleware software support?
- What characteristics aid in the evaluation of middleware software tools?
- How do DOD standards impact the selection of middleware software tools?
- Are applications developed with middleware tools easier to maintain than third generation legacy systems?

B. METHODOLOGY

A study of client/server theory was conducted to provide a basic understanding of this type of information system. A literature search was done to provide insight into current trends in middleware software development tools and their role in open system client/server strategies. These two elements provide the theories behind middleware tools and their use.

To gain a better understanding of how these theories translate into practice, a small project was designed and implemented using a current middleware tool. The project 's goal was to develop an application to allow authorized personnel access to NPS faculty resume information from a central repository. This was completed using one of the newest Rapid Application Development (RAD) tools in the market, Borland's Delphi for Windows version 1.0.

II. BACKGROUND

This chapter is offered to familiarize the reader with the history of client/server systems and why this computing movement is impacting all areas of information technology. This historical look is followed by a glance at where distributed computing is headed. An understanding of how information systems are evolving and the most-likely next steps will provide better insight into the role of middleware software development tools.

The second half of this chapter furnishes the reader with the definition of middleware. As with any new technology, there are many definitions to choose from. It also demonstrates one of the difficulties with studying any new technology. If it can not be clearly defined, it is difficult to understand.

A. CLIENT/SERVER EVOLUTION

In the past, networks were relatively easy to manage because they were usually homogeneous server-centric systems (Lewis, 1995). A question organizations had to answer was which company would serve their computing needs best? When a single vender is used for all system components, the structure is conceptually straight forward but expensive to operate. An organization was locked in to whatever applications were developed for their system. Limiting a manager's ability to quickly respond to new requirements was his reliance on Information Systems (IS) professionals to program changes. Limited computing power forced the IS professional to focus on optimizing central processing unit (CPU) usage than on the user's computing needs.

The introduction of Personal Computers (PC) in the work place brought computing power to the user, technical difficulties to the IS Staff, and a paradigm shift in the information technology field. The end-user was no longer content to struggle with the rigid restrictions of inflexible systems. As the emphasis turned to meeting the user's requirements, IS professionals struggled to provide information to the desktop. For many this meant redundant stand alone systems. This is still true of DOD's current information

5

system infrastructure which is riddled with single-purpose, vertically-structured, inflexible systems (Clancy, 1994). This is not only inefficient, but data inaccuracies are disturbing to managers, users, and customers.

To meet the user's demands and the need to share data, not duplicate it, vendors developed network technology and specialized software applications. This was the beginning of client-centric client/server computing (Lewis, 1995). These have introduced additional complications to the information technology structure and have transferred the interoperability burden from the vendor's technical specialists to individual system developers. Despite these difficulties, organizations are unlikely to return to a homogeneous system after seeing what this multivendor environment can do. Time and location restrictions are no longer acceptable to the user. No matter where the information is or what format the data is in, the user demands immediate access. Open systems client/server is today's answer to increasing user productivity and achieving flexibility in business computing.

Open systems was originally used to describe the goal of a common UNIX operating system. It is now used in conjunction with modular, component-based systems - a true plug and play environment. The Institute of Electrical and Electronics Engineers' (IEEE) Technical Committee on Open Systems describe it as a "comprehensive and consistent set of international information technology standards and functional standards profiles that specify interfaces, services, and supporting formats to accomplish interoperability and portability of applications, data, and people." (Watterson, 1994)

Client/server means many things to many people. From the hardware perspective, the client is usually some type of smart terminal and the server is the computing environment (Schulte, 1994). The two must interact to achieve cooperative processing. From a software perspective, the client has control of the application program and the server maintains the common functionalities that can be shared with other applications (Schulte, 1994). The client represents the program that makes a service request. The

server performs the requested operations and returns any results. A program can act as a client or a server depending on whether it is requesting or providing the service (Adler, 1995).

Each server maintains a client interface that defines those operations that a client may request. A client can only request those services that fall within the scope of the server's client interface. The services a client may request include naming and authentication from the operating system, shared file and printer resources, and shared applications such as database engines, word processing, or e-mail. (Adler, 1995)

These systems have become increasingly more complex and difficult to integrate and maintain. Continually integrating new applications into the scheme has pushed the IS professional to develop unique solutions for integration and interoperability tasks. The difficulties in coordinating the interaction between the client and the server are particularly significant in heterogeneous systems (Adler, 1995).

The computing industry is moving toward collaborative computing based on peer-to-peer networks and client/server is a necessary step. This distributed computing evolution will evolve through the use of object-oriented technology, document-centric software architecture, data warehousing, standards, and the expansion of end-user programming. The progress toward computing nirvana has been broken down into four stages: (Lewis, 1995)

- Server-centric: dumb terminals connected to a mainframe computer that held the database application,
- Client-centric: the current stage of client/server computing,
- Peer-peer: personal computers connected via a local area network (LAN) and/or a wide area network (WAN) to data warehousing servers with federated databases,
- Fully distributed peer-peer collaborative computing: each element in the network can change roles and serve as a client or a server depending on the application and the best use of system resources.

Clancy conducted a survey of senior executives who are involved in their organization's information system strategy planning. Most of these executives believe

middleware development tools will be a very important factor in revitalizing their legacy systems and developing open system client/server structures. The executives' beliefs are consistent with the published literature's statements regarding middleware's place in developing workable open system solutions.

B. DEFINITION OF MIDDLEWARE SOFTWARE

TechGnosis registered the term middleware in 1988 and they defined it as "a piece of software that sits between a data source and a PC application to deliver information over an existing network connection." (Classe, 1994) They contend that if you can see it or display it, it's not middleware. Middleware is the enabler. The middleware environment has outgrown this initial definition. Middleware has also been defined as any run-time system software that is between an application program and the operating system (Schulte, 1994). This broad definition has lead to some confusion about what exactly is classified as middleware software. Using this broad definition, database management systems and transaction processing monitors are considered middleware, while development tools and system management utilities are not because they do not directly support the application at run-time (Schulte, 1994).

Another broad definition of middleware is "a vague term that covers all the distributed software needed to support interactions between clients and servers (Orfali & Harkey, 1994)." This definition comes closer to how commercial vendors are viewing middleware. Categories of middleware in this definition are: (Orfali & Harkey, 1994)

- Service Specific,
- Distributed System Management,
- Network Operating Systems, and
- Transportation Stacks.

As this circle of software matures, a clearer and more narrow definition has evolved. Middleware is the "network-aware system software, layered between an application, the operating system and the network transport layers, whose purpose is to

8

facilitate some aspect of cooperative processing (Schulte, 1994)." More simply put, "middleware is software that resides between the operating system and the application and lets computers in heterogeneous environments easily exchange information." (Barbagallo, 1994) Included in this definition are message passing mechanisms, remote procedure calls, and database gateways.

The purpose of middleware is to eliminate the need for IS professionals to reinvent the wheel each time they need to solve an integration problem. As client/server computing increases, the demand for real-time interoperability and middleware will also increase (Barbagallo, 1994). "In fact, middleware is the single, most important piece of the client/server puzzle (Lewis, 1995)." Clancy conducted a survey of senior executives who are involved in their organization's information system strategic planning. Most of these executives believe middleware development tools will be a very important factor in revitalizing their legacy systems.

Middleware also eliminates the need for organizations to have specialists who understand communications at a low level or from being tied to a particular hardware platform or database management system (DBMS) vendor. The flexibility to change front-ends without changing the DBMS or server side provides another way to avoid proprietary lock-in and lets you keep the desktop architecture of your choice. (Classe, 1994)

For system developers, the advantage is that they need to learn only a small set of commands to accomplish complex tasks. Middleware programs take these simple function statements and maps them to a set of functions. These complex functions can initialize protocols, handle error recovery, issue structured query language (SQL) statements, and return data to the workstation. (Barbagallo, 1994)

It is common for a new technology group to be ill defined and used as a catch-all classification. As this group of software matures and its role in the computing environment is solidified, the definition of and distinction among the various types of middleware will become clear. Whichever definition you use now, it is evident that there are many

different types of middleware and there are distinctive differences among those within each class of middleware. Each middleware tool has its own strengths and weaknesses and the IS professional must match the needs of the system with the application's capabilities (Schulte, 1994).

III. DISTRIBUTED PROCESSING

Distributed processing styles categorize the various ways to separate client and server services. The way a network employs these styles determines, in part, which components will be under the most stress. It also shows how a system designer can shift service responsibilities to ease the burden on a particular part of the network.

Another aspect of client/server design is the distributed processing architecture. The most common is the two tier model but there is a growing trend toward a three tier architectural design. The client/server style and architecture model are key considerations when selecting middleware tools.

A. DISTRIBUTED PROCESSING STYLES

There is no one right way to configure a system; it depends on the nature of the application. Although not mutually exclusive, there are five basic ways to set up a client/server operation. A client/server system may incorporate all five schemes at one time or another. (Schulte, 1994)

Figure 1 shows the five processing styles and illustrates the movement from total server processing to almost total client processing. This diagram also makes it easy to see that client/server networks may use different styles depending on the process and application involved. An example of a network with mixed processing styles is furnished in Figure 2.

1. Distributed Presentation

Presentation services display information and interact with the user through user interfaces and the delivery of information (Colonna-Romano and Srite, 1995). Software that provides either distributed or remote presentation services is sometimes known as frontware or screen scraping products. The X Window System and terminal emulators

11

Figure 1 Distributed Processing Styles
After Ref. (Schulte, 1994)

Figure 2 A network using multiple processing styles

are also in this category (Colonna-Romano and Srite, 1995). These products can simplify a user's work with applications that have onerous interfaces (Schulte, 1994).

This implementation scheme splits the application's user interface between the client and the server. A common example is found in the terminal-mainframe configuration. The server, in this example the mainframe, performs all of the presentation, presentation control, application function, and data management (Kind, 1994). Software that is created to present information on a particular medium and provides no additional functionality falls outside the definition of middleware (Schulte, 1994 and Colonna-Romano and Srite, 1995).

2. Remote Presentation

The remote presentation style stores all user interface responsibilities with the client where it remains separated from the rest of the application. While some on-line transaction processing (OLTP) applications use this type of implementation scheme, the most common use for remote presentation is an updated interface for mainframe applications and databases through the use of 4GLs (Schulte, 1994 and Watterson, 1994). While this is visually pleasing for the user and easy to implement, no additional functionality is provided. The benefits gained from re-engineering the system are not realized (Schulte, 1994).

The presentation application is modified each time there is a change to the application or data structure. Whether this has a big impact on the maintenance depends on the frequency of changes made to the mainframe data and the number of different presentations associated with it. A system that is seldom modified but supports a large number of different presentations can be more difficult to maintain than a system with frequent changes that effect few presentation applications.

Some products that provide either remote or distributed presentation services are coupled with products that enable it to function in a heterogeneous environment, access multiple databases, or collate information from multiple sources. One example is the Brio

13

Technologies and TechGnosis partnership. Brio's BrioQuery provides a user friendly remote presentation front-end that addresses TechGnosis' middleware product Sequelink. Sequelink provides heterogeneous functionality to multiple types of databases. Once the data has been returned to Brio's frontware, it can be viewed in a user configured form. BrioQuery only permits the user to read data and manipulate it locally. Joining data from multiple databases cannot be done. This lack of functionality is the distinguishing feature of frontware type products.

3. Distributed Function

Distributed functionality allows two or more application programs to operate through some intermediary application (Schulte, 1994). For an application developer, this is the most complex of the processing topologies since two complete applications must be developed (Schulte, 1994). The intermediary application is a middleware tool that enables interoperability through a message-based or remote procedure call (RPC) mechanism. One component communicates with another component to request some action be initiated or to relay information. Middleware tools providing this functionality are commonly identified by what communication scheme they use. Action requests are commonly performed by RPCs, message queuing services, or object brokers (Colonna-Romano and Srite, 1995). A trend among commercial database vendors is an increased use of stored procedures. These procedures remain with the database on the server and interact with the client via RPCs. Information relays are completed using some type of messaging scheme (Colonna-Romano and Srite, 1995). The most common of these communication methods are described in Chapter IV.

4. Remote Data Access

This type of implementation uses the client to run the application and presentation logic while the data management for the dedicated database, file server, or print server is

on the server. The client issues service requests such as file retrieval, SQL queries, or print commands to the server. The requested data is sent to the client for local manipulation and presentation. Because of the large amount of data that may be transferred to the client, the client and the server are usually located on the same LAN in the traditional PC LAN configuration (Schulte, 1994). It is also widely used for UNIX workstation LANs (Schulte, 1994). Remote data access is relatively easy to program because there is only one application program. Products that provide remote data access include file sharing services such as Network File System (NFS) and Distributed File System (DFS) (Colonna-Romano and Srite, 1995).

In this configuration, middleware enables data manipulation from remote computers running varying types of operating systems. Standards that specify network communications for remote data access between a client and server have been developed. These standards are: (Colonna-Romano and Srite, 1995)

- ANSI/ISO Remote Data Access,

- SQL Access Group Formats and Protocols (FAP), and

- X/Open Portability Guide.

The most common interface language between clients and remote databases is SQL (Colonna-Romano and Srite, 1995). Even though SQL standards have existed for many years, database applications usually uses a proprietary version of SQL and therefore, dictate and manage the FAP (Schulte, 1994). Consequently, it is difficult to mix DBMSs with other DBMSs or other applications (Schulte, 1994). Middleware tools that interface between client applications and remote databases must provide a driver or a database gateway that is specific to the database or use a standards based definition and manipulation language such as SQL or Open DataBase Connectivity (ODBC) (Colonna-Romano and Srite, 1995). When a database specific driver is not used, the database's enhanced, read proprietary, features may be lost in the translation if they deviate from the standards

on which they are based. Middleware tools that provide remote data access include Netware, Unix NFS, and DBMSs from Digital, IBM, Informix, Ingres, Oracle, and Sybase (Schulte, 1994).

5. Distributed Data Access

The client portion of the application (user related logic) sends a request to the server portion of the application (data related logic) which either fulfills the request from local data or communicates with a database engine at another location (Schulte, 1994). Because the application resides on both the client and the server, database queries can be tailored to transfer only those fields the user requires rather than the entire record. "It is the only one of the five processing styles that requires two segments of user-developed communication (Schulte, 1994)." This reduces the amount of network traffic and places more computing responsibilities on the server. A single, centralized database will continue to be the most desirable design for enterprise-wide databases of record but real-time distributed database schemes are rarely used because of their expense and difficulty to implement and maintain (Schulte, 1994). This complexity is because of their need for such mechanisms as the two-phase commit to ensure data integrity. Developers have particularly avoided them because of their development and management complexity (Schulte, 1994). Although deferred data delivery schemes, such as data replication and data warehousing, are similar to the real-time applications, they have fewer drawbacks and are more likely to be used (Schulte, 1994).

The most visible difference between a distributed data management system and a remote data management system is where the data is formatted for viewing. Because remote data management is optimized for its local task and messaging is brief, this scheme is most practical for wide area networks. The client and server hold only those functions that are required to fulfill their distributed function tasks. Middleware tools that use this processing style include IBM's Advanced Program-to-Program Communication (APPC), Netwise's Transaccess, Open Software Foundation's (OSF) Distributed Computing

Environment (DCE) RPC, and Sybase's Open Client/Open Server. The Gartner Group estimates 75% of client/server systems use remote data access as their processing style. They expect this to change in favor of distributed data access with the emergence of tools such as network-capable 4GLs, distributed transaction processing (TP) monitors, object request brokers and development tools such as Texas Instruments' IEF for Client/Server and Andersen Consulting's Foundation for Cooperative Processing.

B. DISTRIBUTED PROCESSING ARCHITECTURES

Client/server systems operate in a two-tier or three-tier fashion, using either "thick" or "thin" client software (Schulte, 1994). The number of tiers depends on the number of servers a client must deal with to reach the data (Schulte, 1994). Note that this does not mean the number of servers in a client/server system. There can be multiple servers and still be a two-tier architecture.

The designation of thick or thin depends on where the majority of the functionality is located. A thick client maintains such things as DBMS drivers while a thin client software puts the drivers with the server. There is a move in industry to use thin clients to reduce the network overhead as much as possible (Schulte, 1994).

1. Two-tier

The two-tier system shown in Figure 3 is the PC based LAN configuration with the client directly linked to the host data. It is typically used in small environments of less than 50 users and the only option when you must use remote or distributed presentation processing styles. The two-tier approach generally places more stress on the network and will require a large bandwidth to accommodate the users because all database operations and file transfers are performed over the network.
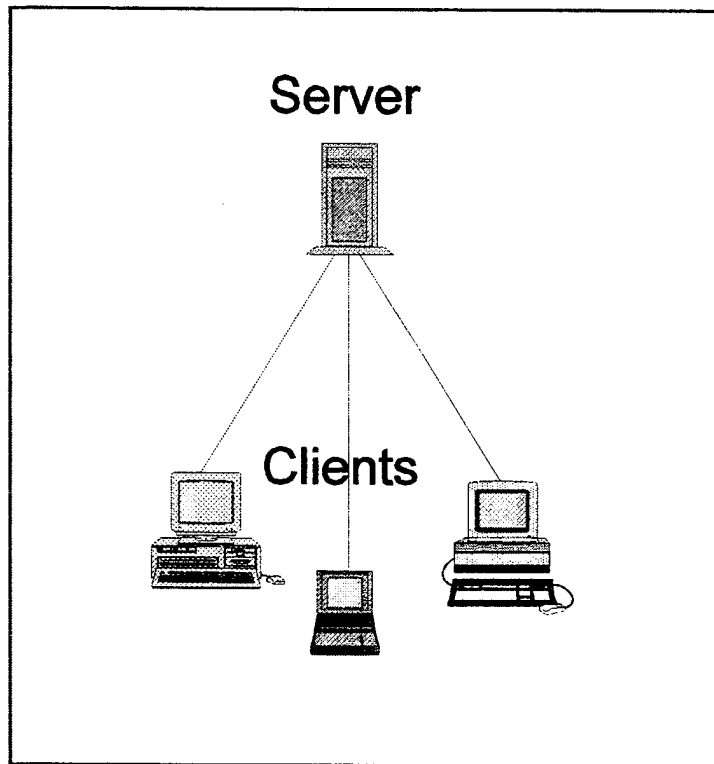
Figure 3 Two-tier Client/Server Architecture

The more advanced form of servers is the database server, transaction server, and application server (Orfali & Harkey, 1994). In two-tier applications such as ODBC or database vendor environments, the client is sending SQL requests as messages to the server and the results of the query are returned over the network. This ties the client tightly with the server because the client must maintain interface resources and be aware of the database's metadata. Databases that support stored procedures help in reducing network traffic by allowing some (usually minimal) functionality (business logic) to be built into the database server. The code that processes the SQL request and the data resides on the server. This allows the use of the server's processing power to find the requested data rather than pass all records back to the client for data filtering.

18

In transaction servers, clients invoke remote procedures that reside on servers which also contain an SQL database engine. There are procedural statements on the server to execute a group of SQL statements (transactions) which either all succeed or fail as a unit. The applications based on transaction servers are called On-Line Transaction Processing (OLTP) and tend to be mission-critical applications which require a 1-3 second response time 100% of the time. These applications also require tight controls over the security and integrity of the database. The communication overhead in this approach is kept to a minimum since the exchange typically consists of a single request/reply. This differs from the multiple SQL statement processing that is associated with the database server.

Application servers are not necessarily database centered but are used to serve such user needs as security, e-mail processing, and network management. Basing resources on a server allows users to share data while security and management services ensure data integrity and security.

2. Three-tier

A three-tier architecture, Figure 4, introduces a server (or an "agent") between the client and the server. The client calls on a local server to perform the interface functions with another server that is usually remote. The middle tier provides translation services (e.g., in adapting a legacy application on a mainframe to a client/server environment), metering services (e.g., acting as a transaction monitor to limit the number of simultaneous requests to a given server), or intelligent agent services (e.g., mapping a request to a number of different servers, collating the results, and returning a single response to the client). One fundamental goal of a three-tier system is to allow for the development of applications which provide users with much more dynamic, up-to-date views of the business. This requires active communications between the clients and the applications server code in the middle tier.

19
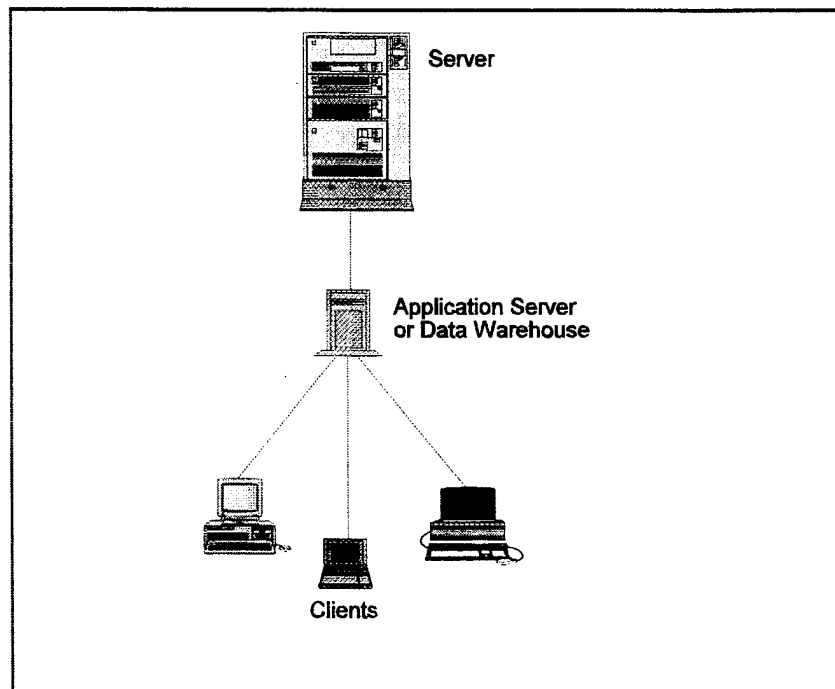
Figure 4 Three-tier Client/Server Architecture

Three-tier configurations are frequently used when connecting users to centralized mainframe databases. The current trend is to replicate the data on the local server. This is done for both security and potential network overload reasons. With direct access, a user could execute a database query that produced an unintentionally large result set. Not only would the user have to wait to see the results before realizing the mistake, but the network response for other users would be degraded as well. The use of data warehouses also allows system administrators to send only the data required by the group of users connected to the data warehouse server. This speeds the system for the local user while protecting the records or data fields that the local users are not privileged to see.

Data warehousing adds system complications that an organization must consider. This is especially true when the users need to be able to modify the data in the database and data changes must be coordinated to assure the integrity of the principle database.

The data that is located at the data warehouse server will not be up to date and it must be updated periodically. These are less of an issue when the data warehouse supports read-only clients such as decision support systems (DSS).

3. Logical Tiers

Client/server architecture is commonly described using physical devices as the determination of whether it is a two or three-tier system. Watterson even described a one-tier client/server system. Although these descriptions are easy for the novice to visualize, their use leads to misconceptions among the uninitiated. Logical tiers are delineated when you have programmatic interfacing between the client/server functions of presentation, business logic, and data access logic. This method of visualizing client/server systems more accurately describes the goal of three-tier architectures.

Using a logical architecture, system integrators and system developers are better able to have a separation of concerns. This principle refers to the partitioning of problems so that two or more parts can be solved independently of one another. Using a logical architecture encourages a more modular design where parts are independently defined and maintained. This separation of functionality also allows each part of the system to be optimized for its particular functionality. The most popular example of this is the separation of the business logic and the database. This can be either a two, left Figure 5, or a three-tier, right Figure 5, design. The client will always control the presentation layer and the business logic layer will always house the functions. In the two-tier configuration, the business logic can reside with the user or the data server.

This logical separation also promoted reuse of business logic. An example of this is an information system which consists of several different applications which share some fundamental data needs such as access to code or methods representing business policies, formulas, or rules. Rather than duplicate the code in each application, it remains separate

21
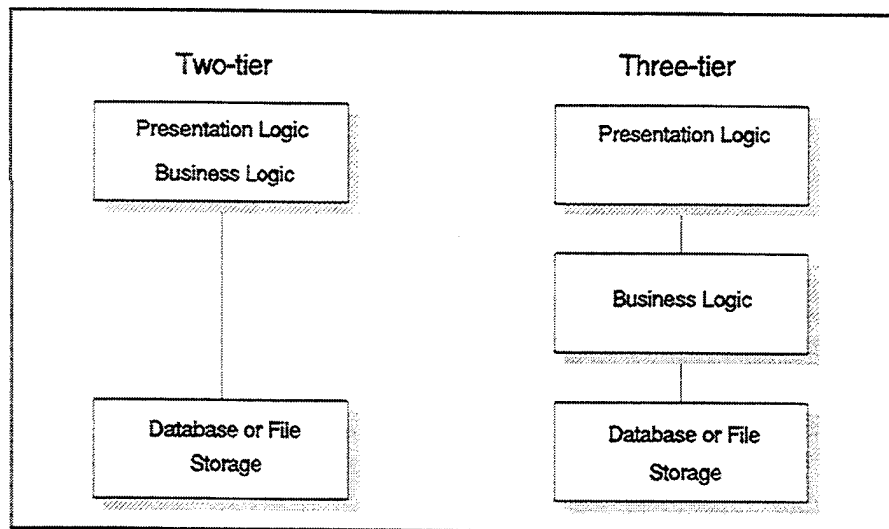
Figure 5 Separation of Business Logic

so that each application can access it as needed. This not only reduces duplication, but requires only one set of updates to be performed when there are changes to the business logic. This example can be expanded to include the centralization of some application code. This allows for those applications that need high-volume access to the business' base data. This is called "application partitioning".

If you use a TP monitor you essentially have three levels:

- The end-user program - The client handles the presentation and does some data checking.
- The TP monitor and the task (server) - The application server which handles the business logic. The transactional RPC middleware is included here.
- The resource, which may also be managed by a server - Resource managers such as relational database management systems (RDBMS), queues and file systems.

Physically you may combine the last two into a single "application" that the client sees. However, SQL is only involved if the resource is a relational database, and it (usually) happens between the TP monitor and the resource -- not the client. The client invokes some kind of service (or task) with a set of parameters. The TP monitor is the

22

agent that takes the request from the client and formats the service request to meet the database requirements. The TP is responsible for the appropriate application program interface (API) to interface with the data resource. TP monitor applications include Encinia Monitor, Application Control and Managment Service (ACMS), and IBM's Customer Interface Control System (CICS). Encinia automatically implements a two-phase commit process to ensure integrity when accesses concurrently by multiple clients.

IV. SERVICE CATEGORIES

Middleware can fill various needs in a client/server environment. Orfali breaks service specific middleware into five categories that match how the commercial market categorizes middleware products. These categories are database, transactional remote procedure call (RPC), Groupware, object, and distributed system management middleware. While these categories are certainly useful, this chapter looks at the services middleware tools provide at a lower level. This is done to provide a closer look at the functional components that are used in middleware tools and how they support other client/server applications.

A. COMMUNICATION

By their very nature, client/server operations are divided across machines, address spaces, networks, and operating systems. It is evident, therefore, that a communication scheme is necessary. Communication exchanges are either tightly-coupled request/reply interactions or loosely-coupled queue-based interactions (Orfali & Harkey, 1994). RPCs are tightly-coupled while messaging is generally loosely-coupled. The network operating system provides most of the low level communication requirements using peer-to-peer communication or some form of RPC middleware (Orfali & Harkey, 1994).

1. Remote Procedure Calls (RPC)

RPCs use non-queuing messages to invoke a specific function that is not part of the same address space (Schulte, 1994, Colonna-Romano and Srite, 1995). This call and return method of communication is similar to procedure calls used in programming. Like procedure calls, all necessary parameters are passed with the RPC. When activated by a process or thread, the RPC function is carried to completion and then control is returned to the sender. Figure 6 depicts the traditional RPC blocking protocol where client

processing is suspended until the results of the RPC are received. A non-blocking

protocol allows the client process to continue while awaiting results. Many vendors have

implemented a non-blocking RPC protocol in their software but this will always be

synchronous communication because this type of middleware does not store messages for

later delivery (Schulte, 1994).
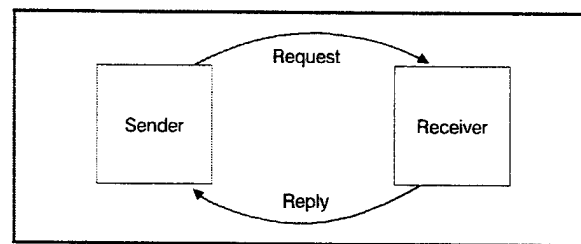
Figure 6 RPC Request/Reply Call

Software developers use RPC services to distribute workload or use common

facilities that are remotely located (Colonna-Romano and Srite, 1995). An application can

use remote procedures the same way it would a local procedure. Available functions and

their parameters are usually defined and passed between the client and server in

compliance with the network operating system's (NOS) network interface definition

language (NIDL) (Orfali, 1994). Stubs are developed using a NIDL compiler and are

linked with the calling code. The client stub packages the parameters in a RPC packet and

passes the packet to the run-time library (Orfali, 1994). The server stub unpacks the

packet, calls the remote procedure, and repackages the results before sending the reply

(Orfali, 1994). OSF has defined RPCs in the DCE RPC specification with the following

major components: (Colonna-Romano and Srite, 1995)

- Client stubs that interface with remote procedures. Client stubs use a service
  library as an intermediary to the remote procedure. Some applications will
  bypass the client stub and access the service library directly.
- RPC run-time service libraries hold the network interoperability functions that
  enable the client to work with remote procedures.
- Application server procedures are what the client is calling.

If there is a failure in a RPC process, the client will usually time out and retry. The server ensures that only one valid request is executed. To do this, the server maintains complete state information until client acknowledgement is received. Security is automatically incorporated into the RPC by the NOS. The type and level will depend on what is available through the NOS and what was specified when the RPC was developed.

2. Messaging

Message Oriented Middleware Association (MOMA) was formed in 1994 and is one of the newest groups in the standardization arena. IBM, Digital Equipment, PeerLogic, Momentum Software, and Covia Technologies are some of the vendor members of this group (Birkhead, 1994). Their purpose is to develop standards that will dovetail with existing standards such as RPC and networked object databases (Birkhead, 1994). On their internet homepage, MOMA lists their goals:

- To assist users, systems integrators, and vendors in receiving the maximum value form message-oriented middleware through education of the user and vendor communities.
- To serve as a concentrator for interoperability and technology requirements, and to influence the appropriate standards bodies.
- To server as an interchange for experiences and ideas related to the development and deployment of client/server and other distributed applications based on message-passing technology.
- To promote functional interoperability between applications built using different message-passing tools and mechanisms
- To reflect the composition of the community of interest in client/server and distributed computing by including users, vendors, and system integrators.

Message oriented middleware (MOM) is designed to provide distributed transactions and message delivery services (Birkhead, 1994). Messaging middleware is useful when you do not want the client and the server to be tightly synchronized or would like to do batch uploading during off peak system time (Orfali & Harkey, 1994). Messaging can be implemented with or without deferred delivery capability. Deferred delivery queues are useful for the nomadic laptop user. In most instances the message will maintain the state

27

information and the sender will remain unblocked (Schulte, 1994). This is not true when messaging is used with RPCs and object request brokers (ORB) (Schulte, 1994).

Messages are usually unstructured or semi-structured data, such as text or images but they can be fixed-format data, such as a record. Messaging can be handled by a single application or layered with other messaging applications, messaging subsystems, or other programs. Messaging middleware generally does not perform any data translation functions. E-mail was the first type of messaging system but has been expanded to other interaction uses, such as program-to-program, program-to-person, or person-to-program (Schulte, 1994). Message-oriented middleware lets application programmers write to a common API. Application designers determine the message format and the actions. This is the same concept as objects that encapsulate logic and data.

3. Message Queuing

Message queuing is based on some type of intermediate message storage with the message maintaining the state information (Schulte, 1994). This type of implementation is analogous to the post office and can be used to create one-to-many or many-to-one relationships (Schulte, 1994, Orfali & Harkey, 1994). The delivery is dependent on the system, the workload, and the class of message. As Figure 7 shows, this is an asynchronous method and the sender is usually unblocked. If needed, message queuing middleware usually supports synchronous blocking message transmission (Orfali, 1994). Specific implementation details are dependent on the application and system requirements, but a universal characteristic is that programs communicate through a queue. This means that programs do not have to be available at the time communication is initiated.

"Message queuing middleware is designed for serious production application and is capable of high throughput rates and fast transfer times (Schulte, 1994)." Middleware that uses this type of communication is commonly used for complex, critical production applications, where there is no tolerance for message losses (Schulte, 1994). Messaging
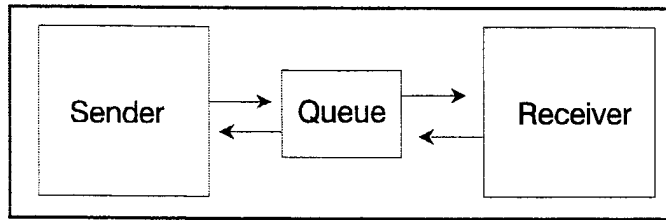
Figure 7 Message Queuing

middleware is available to support a variety of operating systems and network protocols. Transaction message queuing systems have monitoring capabilities that offer the user increased data integrity.

B. CONTROL

Control services are used to invoke applications or control the execution of multiple applications. The applications could be multiple instances of the same application or different applications that need to coordinate their execution. Four types of control services are discussed.

1. Object Request Broker (ORB)

It is easier to develop and deploy applications using ORBs once you under stand ORBs themselves (Schulte, 1994). ORBs are similar to RPCs in that they execute distributed function communication and integration using a call-and-return paradigm (Schulte, 1994). Distributed function communication is program-to-program, or peer-to-peer, communication. System integration projects use object brokers to unite new applications with existing applications in an object-oriented manner but object-oriented systems are inherently message-based (Colonna-Romano and Srite, 1995, Schulte, 1994). "An object sends a message to another object, which executes its methods (processing routines) to act on its associated data (attributes) (Schulte, 1994)." In a networked system, the messages are sent via an object broker. An ORB exploits the characteristics of

29

Object Oriented Programming (OOP) and is the foundation for building distributed object applications (Orfali & Harkey, 1994). An application, which provides a service, registers its objects and operations with the object repository. Applications that need a service select the appropriate operation and invoke an instance of the servicing application which in turn processes the desired service.

Both ORBs and RPCs must associate the caller with its intended receiver. This association is called binding. Dynamic binding is done at run-time and uses the network's directory service to locate its receiver while static binding is coded in to the ORB or RPC. Automatic binding relies on the ORB or RPC's ability to locate the appropriate receiver. Receivers may also be found using some type of broadcasting method. (Orfali & Harkey, 1994)

For this class of middleware to take advantage of OOP's low coupling modularity, it will need to rely heavily on the use of standards. If industry wide standards become a reality, vendors can enhance the performance and functionality of ORBs and this middleware will enjoy widespread use in mainstream applications (Schulte, 1994). The first applications will probably be in system and network management (Schulte, 1994). One such product is Forte development tool. In addition to providing its own type of ORB, it complies with Common Object Request Broker Architecture (CORBA) and Object Linking and Embedding (OLE) object broker standards. This product is also capable of doing application partitioning.

2. Transaction Management and Transaction Processing Monitors

A transaction is a set of operations that are executed as a whole or not at all. The application programmer encapsulates the transactions so they process as a unit. They are commonly thought of as a business event; but they have evolved into a design philosophy synonymous with robust distributed transaction processing (Orfali, 1994). A transaction is characterized by its atomicity, consistency, isolation, and durability properties (Orfali,

30

1994). Viewed from the client's perspective, atomicity is the all-or-nothing execution mentioned above. If it cannot be executed completely, it will return to the client as if no execution took place. This leads to the second property of consistency. After execution, the transaction must leave the system in a correct state or abort (Orfali, 1994). If it aborts, it must return the system to its initial state (Orfali, 1994).

Isolation ensures that a transaction's behavior is not influenced by outside influences. Transaction processing uses serialization to control resource accesses and guarantee that programs running concurrently will not corrupt the transaction's operation (Orfali, 1994). This allows a multi-user program to operate the same as a single user version. Durability and persistence are interchangeable and refer to the permanent effects of a transaction after the two phase commit is positively completed.

The transaction management service uses transaction managers and resource managers to coordinate transaction execution using a two phase commit protocol. The resource manager controls resources, a database for example, and any changes to it. The transaction manager coordinates with resource managers and other transaction managers to complete the two phase commit protocol. The transaction manager is part of TP monitor software and therefore not applicable to all types of middleware tools. Useful questions in evaluating software that provides this type of service are:

- Does it provide support for distributed applications?
- How difficult are transactions to develop?
- Does it comply with applicable standards, such as X/Open and CORBA?
- Does it support transaction execution on multiple nodes or just single nodes?

One of the benefits of TP middleware is relieving the application programmer from developing or including code to execute transactions. The execution logic is the common denominator among transaction processing. The application programmer needs only to develop the business logic.

Another positive aspect is the level of trust that can be placed in TP programs. All programs that participate in the TP must adhere to the underlying transactional discipline (Orfali, 1994). This is why many mission critical systems rely on this type of client/server

31

processing. Tuxedo, Top End, Encinia RPC, and CICS are some of the TP monitor programs available. Transaction processing is evolving to make it more flexible and easier to manage but the basic transaction properties will remain (Orfali, 1994).

In the same category are OLTP client/server programs. OLTP breaks the traditional transaction into steps using short transactions that execute sequentially (Orfali, 1994). This reduces the number of failures experienced by large transactions and does not monopolize critical database systems.

3. Multi-threading

A thread is the sequential flow of control in a program. A single thread has a single point of execution. In a multi-thread program there are multiple threads that are executed concurrently with a corresponding number of execution points. These threads share a single address space. Synchronization operations provided by the operating system ensure memory is correctly addressed. (Colonna-Romano and Srite, 1995)

Multi-threading is an enabling technology that is implemented by operating systems such as OS2, Windows NT and Digital's OpenVMS (Colonna-Romano and Srite, 1995, Orfali & Harkey, 1994). These operating systems allow applications to take advantage of the benefits of multi-threading. The benefits include improved user interface responsiveness; servers that can handle service requests from multiple clients; and increased program performance through improved throughput, computational speed, and responsiveness (Colonna-Romano and Srite, 1995). Server programs support multi-threading so it can service multiple clients quickly and safely. Middleware tools that use multi-threading are able to take advantage of parallel processing and multi-tasking environments (Orfali & Harkey, 1994). This contributes to the scalability of the client/server model (Orfali & Harkey, 1994).

4. Continuous Computing (Fault Tolerance)

Sophisticated systems are resilient to failure. Agent software that carries out defined tasks without intervention should retain state information so that tasks can be completed when a system failure is repaired (Palaniappan, 1992). Their resilience is judged by the degree to which they are able to cope with and recover from system or network failure (Palaniappan, 1992). Some degree of resilience is built into application programs, such as TP monitors or timed backup services. Forte has a fault recovery scheme called failover that uses its ability to clone objects and run multiple copies to preserve application processing functionality by switching to another machine if the primary machine goes down.

For more sophisticated fault management, third party fault management and help desk tools are available. These programs receive alarms, identify the failure, and launch corrective action (Orfali, 1994). They can be an integral part of the information system, a single focused add-on tool, or included in a comprehensive network management tool. Their primary purpose is to span all areas of the network, database, and system management functions to aid the organization's system manager in correlating failure symptoms (Orfali, 1994).

After an alarm is activated, the tool should log the error and notify the appropriate people by a mechanism, such as e-mail, beeper calls, alarm windows, or flashing icons. The more sophisticated tools provide scripting facilities that can be used to design repair scripts for common errors. The custom script should be able to execute other programs to solve the system error or gather additional information. When the tool can not fix the error, the fault management tool can narrow down the areas of concern through a filtering process so the system manager can focus his or her attention to the most probable areas.

C. INFORMATION SERVICES

The primary purpose of client/server computing is the centralization of common data and resources. There needs to be a way to control the users and their application's use of these elements to ensure they are available throughout the system. Information and system resources are shared among users through information services.

1. Data Access

Database middleware grew from the need to access and update legacy or SQL data and is still one of the primary uses of this type of middleware (Birkhead, 1994). Data access services enable applications to access data in relational database tables across a heterogeneous network environment. Besides updating legacy databases, this group of middleware can enable access to multiple DBMS engines, add functionality to existing DBMS products, or assist in program development (Schulte, 1994). There are basically two methods to implement database access: (Stodder, 1994)

- bring the query to the database (data warehousing and remote data management) or
- send the data to the user (data replication and distributed data management).

Right now, replication and on-line analytical programming (OLAP) are the popular choices because it is easier to implement and maintain (Stodder, 1994). Legent (LMP/XP) and Evolutionary Technologies (ETI Extract Tool Suite) are two of the more prominent companies in this class (Stodder, 1994). A draw back of this type of implementation is the amount of data that may be transferred across a network.

The most sophisticated form of DBMS middleware is the database gateway. Sybase's Enterprise Data Access/Standard Query Language (EDA/SQL) and Micro Decisionware's Database Gateway are examples of gateway products. Database Gateway is a general purpose gateway based on Microsoft's Open Data Services (ODS) server development platform. Whether to use a direct connect two tier architecture or a three

tier gateway approach will be determined primarily by the current system and the database involved. A three tier approach is appropriate when there are multiple protocols involved, there is no two tier driver available, or when the use of a two tier driver is too costly in terms of system resources or price (Microsoft, 1994).

The most commonly used database definition and manipulation language is SQL (Colonna-Romana and Srite, 1995). SQL based middleware products work well for decision support (read-only) access into one or more relational databases managed by the same DBMS product (Schulte, 1994). It is less successful across heterogeneous databases, especially when attempting to use non-relational data (Schulte, 1994). This class of middleware can: (Schulte, 1994)

- translate SQL into non-relational data manipulation language without user programming;
- maintain a global directory and optimize requests that require access to multiple systems;
- perform joins, unions and other operations across multiple, dissimilar back-end databases; and
- support an RPC mechanism for user developed custom code.

SQL based products can support simple data items, such as character strings, numbers, and dates. Some support more complex data, such as audio, video, and compound documents. Data interfacing is provided through the two interface languages: data definition language (DDL) and data manipulation language (DML). The definition language is used by the database developer to create, modify, and delete the database schema. DML is used by both developers and end users to perform standard data manipulation or relational operations. (Colonna-Romana and Srite, 1995)

2. File Sharing

Remote file systems provide different functions than remote database management systems. A remote file system may be used to enable sharing of large objects such as computer-aided designs or digital medical images or to provide a copy of a software application only when needed at run-time. This saves remote computers from storing

35

large files that are common among the users. This also allows the person responsible for software maintenance to centrally update software eliminating the need for individual installation. Remote file systems allow users to check out the whole object, revise it, then check it back in. When needed, a read only restriction can be placed on the objects. Remote file systems may also be used to describe a method of allowing real-time database access and update with record lock out features to ensure data integrity. (Schulte, 1994)

Originally, network file services were provided by the NOS using logical drives. The NOS enabled various operating system (OS) platforms to view files and share resources the same way they would with their native files and resources. With the development of global naming standards, network file services will provide more functionality across more platforms more easily. The distributed file service defined in the DCE standard includes a naming service that is location independent and integrates file security mechanisms. It is designed to work with existing file servers, such as NFS and the UNIX file system. It also supports file replication and a transactional log that assists in bringing the system back to a consistent state after a crash.

3. Repository

Repository services facilitate information sharing by storing the information's metadata (Colonna-Romano and Srite, 1995). The repository can also store version, configuration, and file management information and compound document information (Colonna-Romano and Srite, 1995). This is useful when there are multiple users working with a set of files associated with a project. The repository service brings order to a file sharing environment. According to Colonna-Romano and Srite, the repository has an object-oriented information model and provides the following features:

- change management,
- version management,
- configuration management, and
- file management.

36

Change management is one of the fundamental services a repository provides. When a file or database's structure is changed, the repository service sends a change notice to all elements affected. When this service is supported, the application can receive the notice and issue a warning to its users. Otherwise the user can review the change notices manually and take the appropriate action. These change notices are discarded by the repository when the affected object is updated. (Colonna-Romano and Srite, 1995)

Version management controls access and tracks how objects were changed. This solves many of the problems that arise when multiple users share files. This part of the service forms the basis of configuration management that allows users to group objects into any useful configuration. The configuration manager supports multiple versions of the same object. (Colonna-Romano and Srite, 1995)

File management tracks the location of files located in and out of the repository. This enables objects in the repository to reference files that reside outside the repository. (Colonna-Romano and Srite, 1995)

4. Directory

Network elements and applications need a centralized source to find information and the directory service provides it. The most recognizable directory service is that which is provided by the International Telephone Union-Telecommunications Standardization (ITU-TS) standard X.500 that provides information for X.400 e-mail services. In this example, the directory holds all user names and their network address. Directories are also used to store information about other network resources. A resource directory might store the current address and interface version number for all network printers and servers (Colonna-Romana and Srite, 1995).

The directory does not necessarily contain the most current information. This is can be the cause of Internet Domain Name System (DNS) lookup failures. Some directory servers, like the DNS, can be configured to accept updates only from the master directory. (Colonna-Romano and Srite, 1995)

37

V. EVALUATION CHARACTERISTICS

Picking the correct development tools is one of the hardest and most important software selection decisions. Ideally, an organization will select several and evaluate them by a parallel development process (Watterson, 1994). Relying on vendor claims of mission critical support is not enough. It is important to select tools that fit both the technical needs and organization culture (Watterson, 1994). Organizations will have to weigh these evaluation characteristics according to their particular needs.

A. USABILITY

"Usability is the extent to which a system or component helps users get their jobs done efficiently and effectively (Colonna-Romano and Srite, 1995)." The primary purpose of an information system is to improve the productivity and effectiveness of its users. Therefore, usability is the most important of all the evaluation characteristics (Colonna-Romano and Srite, 1995). With that said, usability is also one of the most subjective of the evaluation characteristics. Usability means different things to different people and it is based partially on other attributes such as integration and performance (Colonna-Romano and Srite, 1995).

Included in this evaluation characteristic is the learning curve required to become efficient and productive in the tool's use. The time required must be included in all budget and development time calculations. The actual learning time depends more on the tool's user, their background, and their motivation level than on the complexity of the application (Watterson, 1994). Application developers take between one to six months to master the use of a new development environment (Watterson, 1994). Whether or not a vendor supports this learning period through intensive training classes will be a consideration for an organization when making their tool selection. Keep in mind also that the developer may not only need to understand the development tool but the applications they must interface with as well (Watterson, 1994).

39

B. DISTRIBUTABILITY

Distributability is a measure of a service, tool, or application's ability to execute across multiple hardware components of a network (Colonna-Romano and Srite, 1995). This measure can range from none (single user, single OS platform) to complete distributability (any number on users, any OS platform, any network) (Colonna-Romano and Srite, 1995). For most systems, complete distributability will not be required nor will you find a single application that supports complete distributability. Middleware distributability is based on the following: (Colonna-Romano and Srite, 1995)

- Transparent distribution of services. The client does not need to know if a service is distributed or the location of the server. The system developer can design, construct, execute, and manage components without special regard to where the components are located. Distribution should be transparent to both the application and the application development environment.
- Multiple distribution models. The software should support either two or three-tier distribution models.
- Flexible distribution styles. The software should allow developers to use any or all of the distribution styles discussed in Chapter 3. Their use should be allow either sequentially or simultaneously.
- Immediate and queued operation. This enables clients and servers to be distributed in time as well as space.
- Distribution over all networks. The software supports LANs, Metropolitan Area Networks (MAN), and WANs and single or multiple processors.
- Distribution in all life cycle phases.

C. INTEGRATABILITY

"Integration is the ability of applications to work together in a consistent manner to perform tasks for the user (Colonna-Romano and Srite, 1995)." Middleware's integration is based on its interoperability, the extent to which components invoke and exchange data effectively, and uniformity, the extent to which components are consistent with respect to a set of attributes (Colonna-Romano and Srite, 1995). This leads to the first and most obvious question. Does it work with the system's current applications and

databases that are to be retained in the redesign effort (Watterson, 1994)? If the answer is no, then no further evaluation is necessary.

In an ideal world, all components in an open system client/server environment would be highly integrated and interoperate seamlessly. On a practical level, information systems use components that are built with multiple technology bases using different interfaces (Colonna-Romano and Srite, 1995). This real world dilemma is not new. The very existence of the middleware category of development tools attest to the pervasiveness of interoperability issues. Several techniques are available to achieve integration with legacy software:

- Gateways provide communication integration through communication protocol translation. Common types include database and e-mail gateways (Colonna-Romano and Srite, 1995).
- Converters translate between representations and are commonly embedded into gateways (Colonna-Romano and Srite, 1995).
- Encapsulation hides the implementation details behind a common interface (Colonna-Romano and Srite, 1995).
- Vendor specific drivers are written specifically to interface with a given application or database.

The drawback of using gateways, converters or encapsulation integration techniques is the loss of some program functions. These techniques usually support only a common subset of functions between the applications (Colonna-Romano and Srite, 1995). While vendor specific drivers support more of an application or database's unique or proprietary functions, it may lose in other areas such as network communication optimization. Therefore it is important to understand how connectivity is achieved (Watterson, 1994).

Of prime concern in an open architecture network, is the seamlessness of the program. User acceptance is contingent upon a program's ability to work within the tools that a user commonly uses (Palaniappan, 1992). This requires an interface that is consistent with the user's standard environment (Palaniappan, 1992). There should be a degree of uniformity with a user's computing environment. This adheres to the principles

41

behind successful operating environments such as Windows. Each application has a similar look and feel reducing the user fear factor that accompanies the introduction of new applications.

For middleware, consistency and uniformity go beyond the user interface. It includes APIs, system interfaces, and communication interfaces (Colonna-Romano and Srite, 1995). For the developer, being able to write to a single API in a heterogeneous environment is a useful feature. Uniformity reduces complexity and results in a system that is easier to maintain. In an organization that is prone to staff turn over, this is extremely useful.

D. CONFORMABILITY TO STANDARDS

Most middleware uses either de jure, de facto, or other established industry standards (Colonna-Romano and Srite, 1995). What standards a middleware application supports and how closely it follows those standards is an important consideration for the system integrator. Having products that support the same standard does not ensure compatibility however. This fact is quite evident when you examine incompatibility among the many database applications that claim compliance with SQL standards.

The first step in achieving an open system and evaluating compliance to standards is an examination of the organization's own internal standards. Developing a strategic standards profile for the organization provides application and development tool evaluators essential information. The profile development process itself serves as an organization-wide review of where it is and where it wants to go. In doing the review, the organization may find that they have conflicting standards deployed in various places. Recognizing this early in the enterprise planning will reduce interoperability problems later. At the very least it will draw attention to the potential problems so a solution can be reached.

E. EXTENSIBILITY

Extensibility is the degree to which a user or system programmer can adapt and enhance a system's or agent's functionality to meet new requirements (Palaniappan, 1992, Colonna-Romano and Srite, 1995). This includes the ability to add or modify a function, data type, file format, database schema, or information model without requiring changes to existing functions, data, and interfaces or introducing unwanted side affects (Colonna-Romano and Srite, 1995). Unwanted side affects include degradation of performance, reliability, or maintainability. There are three types of extensibility: (Colonna-Romano and Srite, 1995)

- Customization allows end users to made modification to the look and feel of the user interface.
- Configuration adaptability lets the system integrator to make site specific changes to the system and its components to accommodate requirements changes and platform requirements. This includes standardization of components, such as user interfaces.
- Evolution extensibility permits application developers to make internal component changes, such as the underlying database.

Where agents are able to work with other agents to complete complex tasks, extensibility is accomplished through interagent communication (Palaniappan, 1992). This capability enables system developers to create a decentralized architecture. Decentralization makes a system more manageable because extensions and upgrades are accomplished through module changes rather than changing a single monolithic program (Palaniappan, 1992).

F. MANAGEABILITY

System managers need a way to economically configure monitor, diagnose, maintain, and control computing resources in an information system. "Manageability is a critical cost factor in the operation of heterogeneous, enterprise-wide distributed information systems (Colonna-Romano and Srite, 1995)." The middleware tool's ability to

43

control the computing environment resources consistently and easily is a measure of its manageability. In a network, this implies the ability to manipulate system components remotely and has spawned a growing commercial realm of software tools. The goals for middleware manageability includes the management of all physical and logical components, software, and databases. These management capabilities should span the resource's life cycle and support configuration management, fault management, performance management, accounting, and security.

G. PERFORMABILITY

Software performance is commonly measured solely by its response time. When viewed in the software vendor's environment, generally the software is in optimal conditions and never in the conditions that are unique to your system. Evaluating demonstration versions is useful, but the crippling restrictions built into it will mask its true performance when installed into a full operation. This is be especially true of software that lets you network a small number of users. The performance factors will change as the number of users in the system increases.

While 4GL code is easier to develop, it is an interpreted language and therefore slower than procedural code. If an information system has a lot of 4GL code, response time may be improved if the 4GL code is converted to a 3GL language. The biggest improvement will be seen in systems that run older PC's as clients. A lot of 4GL code running on a 286 or 386 machine will be unbearably slow. This leads to another point. When evaluating software in-house, include one of the organization's low-end systems as well as its high-end systems. The practice of testing the extremes is common in software development and is a good disciplined approach in all evaluation exercises. If the software does not produce acceptable results on an organization's older equipment or requires some type of system upgrade, such as increased random access memory (RAM), the cost of the upgrades must be included into the cost/benefit analysis.

Therefore, performance evaluation should also include its use of resources. Resource utilization and response time provides a well rounded view of software performance. The definition of performance then becomes the software's ability to produce timely results with acceptable costs (Colonna-Romona and Srite, 1995).

Another common performance factor is throughput. This is a measure of the number of completed operations in a given time. This measure is especially important for transaction processing software. Transaction processing software has load balancing capability that will increase its throughput if the resources are available.

H. PORTABILITY

Portability is the ease with which a middleware development tool can be moved from one platform to another (Colonna-Romano and Srite, 1995). "Minimizing the time and expense require to get the application ported to the desired platforms is a fundamental concern (Frank, 1994)." In a heterogeneous environment, a high degree of portability is crucial. Both the initial development effort and the maintenance effort must be considered when evaluating middleware. This evaluation also cannot stop at evaluating the impact based on the present structure; an organization must also look to the future when choosing cross-platform middleware. The system designer must consider how well the software can handle changes to the application or even the operating system. This is not necessarily a point for disqualification, but the organization should know this up front.

Vendors may support multiple platforms with a single program or have a different version of its software for each platform it supports. It is not prudent to rely on a vendor's promise of future platform support. The promised platform support may not emerge or not be developed in the time frame that is important to an organization. Middleware supports portability by providing standards based languages, APIs, and protocols.

I. RELIABILITY

Software must produce correct output on repeated trials. This includes the availability of the system to process information when needed. Reliability is commonly measured as the mean time between failures (MTBF) (Colonna-Romano and Srite, 1995). Another measurement, that can be more useful in judging reliability, is availability. This is calculated by the equation $\frac{MTBF}{(MTBF-MTTR)}$, where MTBF is the mean time between failures and MTTR is the mean time to repair (Colonna-Romano and Srite, 1995). This measurement accounts for the difficulties that can be encountered when diagnosing and repairing a system component that has stopped responding. This will also allow the organization to factor in the probable response times if the vendor or some other outside resource must be called in to repair the system.

Reliability is also judged on an application's ability to protect itself from other applications or elements that stop responding. Microsoft's OLE2 is one object oriented technology that allows application programmers write client applications to protect it self. It also lets programs to use trusted object to increase performance.

J. SCALABILITY

Given a new technology, users will find uses for it that the developers never envisioned. This is a common theme when reading about experiences in networking and client/server computing. It emphasizes the importance of looking at a product's scalability during the evaluation process. Scalability is the product's ability to solve both large and small scale problems (Colonna-Romano and Srite, 1995). A common error in client/server development is to prototype an application in a small, two-tier environment then scale up by simply adding more users to the server. This approach will result in over whelming the server. To properly scale to hundreds or thousands of users, it is usually necessary to move to a three-tier architecture. This is why it is necessary to look at the middleware

development tool's scalability and why it is usually a good practice not to install a product whose maximum limitations meet your current size constraints.

It is sometimes desirable to use a tool such as Smalltalk to produce a fully working prototype quickly even though it cannot be scaled up to meet the organization's needs. This enables developers to concentrate on the business logic rather than the development environment. It also provides valuable user feedback before extensive time and effort has been expended on a large scale version. After user acceptance, application development using more sophisticated tools or procedural languages is completed in much less time because the prototype serves as a detailed program design. One drawback that may be encountered is mapping tool specific constructs that were used in the prototype to the procedural language or development tools development language. More time is spent getting the requirements correct which will benefit the project in the long run.

VI. STANDARDS

Although DOD spends an estimated $10 billion annually to operate and maintain its AIS application software, the lack of standards prevents correct data retrieval due to the lack of data and metadata standardization (Clancy, 1994). The use of proprietary software and hardware makes it difficult, or sometimes impossible, to interoperate information systems (Aiken, Muntz, Richards, 1994).

A. COMMERCIAL STANDARDS

There are no standards that specifically address middleware software development tools. Middleware products implement services that are defined by some set of standards and there are numerous commercial standards that the system integrator must take into account when developing client/server systems. The three terms associated with standards that are commonly used in the commercial market are: (Colonna-Romano and Srite, 1995)

- De jure standards. These are standards that are created by formally recognized standards developing organizations, such as the International Organization for Standardization (ISO) and IEEE. What distinguishes this category of standards development is their use of the rules of consensus in an open forum. Because of their formal development process, they are sometimes referred to as formal standards.

- De facto standards. These standards have become standards based on their wide spread use in the market place. MS-DOS is one of the most easily recognized member of this group. De facto standards are commonly referred to as industry standards.

- Specifications. This describes features that a software or hardware compliant item will possess. Although not a standard, they are often incorporated into or become standards.

An organization's information system will function more smoothly and the integration of new applications will be accomplished more easily when standards (de jure

49

or otherwise) are used. While formal standards are preferred, they do not always exist or there are conflicting standards being supported by the market's major developers. When this happens, an organization must rely on their internal profiles or guidelines to form the basis of their information technology selections and future development plans (Douglas and Ghoshal, 1995).

The software industry will respond to rising interest and complexity in client/server computing by forming consortia and back-room alliances (Lewis, 1995). This will not be because of some desire to serve the computing community, it will be their desire to capture market shares. These alliances may change frequently as companies reassess the market trends and realign themselves to what they perceive as mainstream computing. These alliances influence the development of standards and by their support, determine which specifications become industry standards.

This research was not designed to present the reader with a complete review of all standards. It is also not realistic to expect an IS professional to possess an in-depth knowledge of all standards. It is important however, that IS professionals be familiar with the more important standards in use and those emerging standards that are set to dominate the commercial market place.

The trend in today's development market is toward object oriented (OO) technology and the most technical challenge is how to share objects across a network (Watterson, 1995). There are standards that are emerging that the backing companies hope will dominate the software industry. This paper will provide a summary of the following two object oriented standards:

- Microsoft's Component Object Model (COM) and Object Linking and Embedding 2.0 (OLE2) and

- Object Management Group's (OMG) CORBA.

The other standard presented is OSF's DCE. It is establishing itself as an important standard in the client/server market (Watterson, 1995).

1. Distributed Computing Environment (DCE)

In 1991, the first major proposal for an industry-wide middleware infrastructure standard was announced by the OSF consortium. OSF's DCE was intended to accelerate the deployment of heterogeneous, networked applications by using a collection of middleware services (Schulte, 1994). DCE claims to be the only suite of standards that covers the distributed application environment (Colonna-Romano and Srite, 1995). DCE consists of multiple components which have been integrated to work closely together. As illustrated in figure 5, the major components of DCE are: (OSF, 1992, Orfali, 1994)

- The Cell and Global Directory Services. Directory services provide file location services using an extended version of the ISO directory service provided in X.500 global directory services. DCE's Global Directory Service is based on Siemen's DIR-X product. Its extended functions provide added protection and security to data stored in the directory, directory replication for increased reliability, and sophisticated caching techniques for increased performance.

- Security Service. Security services provide a means of resource access control throughout the distributed environment. Using access control lists, it allows users or groups of users to be assigned document access rights. DCE's security services are a superset of the POSIX ACL draft standard 1003.6.

- Distributed Time Service. Timing services synchronize clients and servers which aids in DCE's multi-threading abilities.

- RPC - allows a locally running application to call procedures that are available on a remote computer.

- DCE Threads. Thread support allows applications that support this feature to run on several computers concurrently. It is based on Digital Equipment Corporation's implementation of Concert Multi-thread Architecture (CMA). A multi-threading API also allows non-blocking RPC's to be issued. It permits programmers to use the fork and wait process programming model which allows for RPC's to be issued to different computers concurrently and synchronization of the results.

51

- Distributed File Service (DFS). DCE's file service is based on Andrew File System from Transarc and the diskless client from HP. DFS provides a uniform name space, file location transparency, and high availability. DFS's APIs are based on POSIX 1003.1a and is interoperable with Sun Microsystem's NFS.
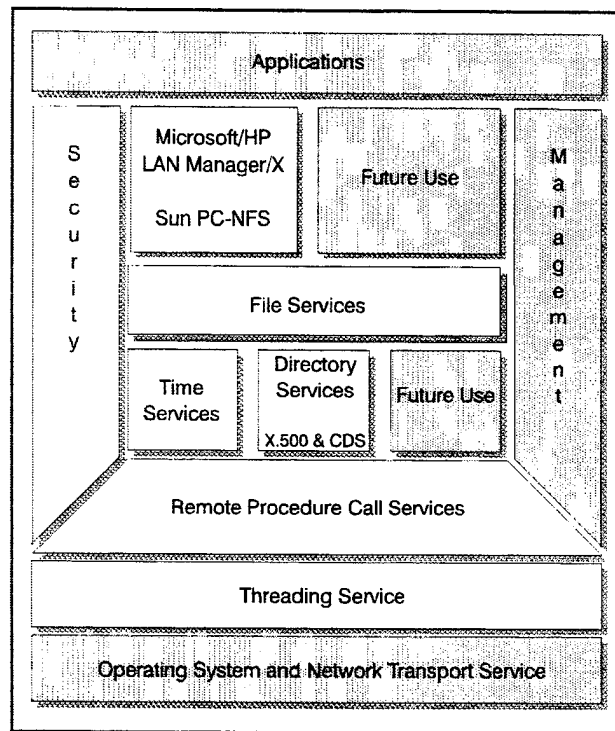
Figure 8 OSF's DCE
Reproduced with permission of the copyright owner

The Threads, RPC, Cell Directory Service, and Distributed Time Service components are commonly referred to as the "secure core" and are required components of any DCE installation (OSF, 1992). The Distributed File Service is an optional component but is integrated with the DCE security and RPC mechanisms. Its integration and use of the other DCE services, it can be administered from any DCE node (Orfali, 1994) Although it chose strong technologies, it was not adopted by the rest of the industry in its entirety (Schulte, 1994). Of the five core components it identified, the

52

GartnerGroup predicts that only the security and naming services will be widely adopted. This is because there are no viable, multi-vendor alternatives (Schulte, 1994). The other components will have varying degrees of support within the industry (Schulte, 1994). One of the most notable company providing industry support is Digital Equipment Corporation.

OSF calls DCE an enabling technology and categorizes it with middleware. It was not developed as a stand alone environment. It should be integrated or bundled into a vendor's operating system. The DCE components can replace the vendor's current components which fulfill the same function but do not support network functions. When compared to ISO's Open Systems Interconnection (OSI) seven layer model, DCE's RPC uses layers 5, 6, and 7. DCE works with OSI layers 1 through 4. OSF's directory services enable clients to request services without knowing the specific server, its location, or the exact name the server has given the service. The operation is invoked based on a description of the requested service. Their implementation is very similar to what ISO adopted in their open distributed processing (ODP) enterprise reference model. (Adler, 1995)

2. Component Object Model (COM) and Object Linking and Embedding 2.0 (OLE2)

Any technology that Microsoft develops and/or supports must be given careful consideration. Microsoft not only influences all the commercial software industry but the scientific community as well (Lewis, 1995). OLE is Microsoft's architecture for desktop document-centric computing and COM is its corresponding implementation layer (Lewis, 1995).

In Microsoft's world, clients are primarily users of interfaces implemented on other components and servers are the modules that implement components with interfaces. These elements are based on COM and are called "Compound Objects" because they support the basic object oriented properties of reusability, encapsulation, and

53

polymorphism (Microsoft, 1994). OLE and COM were developed to aid in revitalizing legacy code without requiring major re-architecturing or rewrite efforts (Microsoft, 1994). This does require a system designer that is savvy enough to recognize portions of their legacy code that is encapsulated already or features that would benefit from using other objects. Once identified, OLE and COM interfaces are layered on top to enable integration.

OLE2 uses COM for low level communication between components through a binary standard of interfaces. It also uses native operating system APIs rather than replacing them with redundant APIs. This API exploitation helps OLE to integrate applications running on the operating system. One of the more interesting and useful features of this technology is functionally evolving objects. This is the ability to deploy new or updated objects into an existing system without altering existing system components. (Microsoft, 1994)

OLE2's structured storage, naming standards, and standardized stream formats allow data to be extracted from files that are not native to the application. For now, this is restricted to file information and document keywords contained in the documents's summary information. Windows 95 will take advantage of this to allow users to search all files by their keywords, origination date, or author. (Microsoft, 1994)

OLE supports object persistence with each object retaining its own state information. This reduces the overhead of compound documents. OLE also provides for object automation. Through such development tools as Visual Basic, an application developer can develop programs with encapsulated business logic to create and manipulate automation objects. These custom components can be accessed by higher level tools, 4GL languages, or application macros to achieve interoperability. (Microsoft, 1994)

3. Object Management Architecture

In an effort to bring order to this sector, the OMG vendor consortium was formed in 1989 to develop ORB API standards. The vagueness in which they wrote their first published standard, CORBA, left so much to interpretation that two CORBA compliant products would not interoperate. CORBA 2 was due early in 1995 and was to be specific enough to foster interoperability among vendors. (Schulte, 1994)

The directory services described in section 1 above influenced OMG's work on CORBA. "The CORBA specification adopts an object-oriented model for organizing and managing constituent elements of ODPs - utilizing the close relationship between object interfaces (which encapsulate object methods' behavior and implementation) and client interfaces (which conceal servers' internal structure and behavior) (Adler, 1995)."

One company that is using OMG's efforts is Component Integration Laboratories (IDL). They developed OpenDoc in response to Microsoft's OLE and Distributed System Object Model (DSOM) to Microsoft's COM. IDL uses the CORBA IDL specifications for its System Object Model (SOM)/DSOM stubs. (Lewis, 1995)

The OMB specified a standard interoperability RPC protocol, Internet Inter-ORB Protocol (IIOP) which is not DCE's RPC but they go on to describe a standard of how to do inter-ORB messaging using the DCE RPC in the DCE environment specific inter-ORB protocol (ESIOP). This gives software developers the option of not being CORBA 2 interoperability compliant but providing inter ORB communication via the DCE RPC interoperability standard. They could also choose to provide both communication schemes. This reduces the competition between the DCE and CORBA worlds.

B. DOD STANDARDS

DOD developed the Technical Architecture Framework for Information Management (TAFIM) document to structure the spread of information technology from the mobile combat arena to the fixed office environment. It calls for adherence to a set of

interface standards to facilitate software use in distributed systems. It also calls for rapid application development (RAD) to support continuous process improvement efforts. The goal of RAD and prototyping during the system development cycle is a reduced development time from months and years to days and weeks. (Clancy, 1994)

Commands are tasked with using commercial software products whenever possible. To enhance the user's productivity and innovative initiatives, users will be provided customization tools for tailoring menus, screens and applications. Through training and the use of new tools, DOD hopes to create technically literate users that are able to perform simple modifications, such as configuring or modifying an interface to meet their needs. (Clancy, 1994)

TAFIM is based on an open systems environment and the use of both de jure and de facto standards (Clancy, 1994). DOD has adopted ISO's Open Systems Interconnect (OSI) model for distributed computer communication services. The OSI model addresses two problems encountered when developing distributed computing systems:
Defines standard protocols to promote industry-wide interoperability andEstablishes standard application services.

C. IMPACT OF STANDARDS ON SOFTWARE SELECTION

"When the pressure is on to deliver new systems, taking time to evaluate the tools needed to build them can seem a frustrating waste of time. Even when you know that an unwise choice can spell disaster (Vowler, 1994)." To put a pretty front end on your system may be another tempting choice, but that cannot make an inadequate system suitable for increased needs (Lindholm, 1994). Reusability must be a major concern when choosing middleware or you will continue to build legacy systems and spend up to 50% of your budget on maintenance (Vowler, 1994).

"If vendors could agree on a common FAP, users would be able to by their client middleware and server middleware from different vendors." Unfortunately, this does not

look like it will come to pass. "More than 60% of client APIs and 80% of client-to-server FAPs for data management middleware during the next five years will be non-standard, proprietary specifications." "High-performance and complex applications will continue to use non-standard interfaces that are beyond the capabilities of plug-and-play techniques." (Schulte, 1994)

Users must consider compatibility issues when purchasing DBMS engines, middleware, and client software. For the time being, this will mean purchasing them from the same vendor. (Schulte, 1994)

Functional overlap within the different categories of middleware will continue to increase, but there will continue to be distinct boundaries between them. Because it is becoming disadvantageous for tool providers to build proprietary cores to their applications, many 4GL, DBMS, and other tool vendors will replace their proprietary low-level communication functions by outsourcing to companies that provide a more standard, generalized technology. The higher level transport functions will remain proprietary though. Do not expect that there will be one middleware application that will fit all domains or that there will be a stabilizing force that brings a convergence in this field. (Schulte, 1994)

Open standards are important to both customers and vendors. If a company's proprietary infrastructure becomes an important piece of the client/server architecture, all other elements of the system will pay the price. For vendors who wish to develop applications that must interoperate, they will have to pay in royalties or development constraints. System developers who implement proprietary software will pay by having their circle of software choices limited by the proprietary software. When the circle of vendors is small, the organization may pay a premium for interoperability.

Standardization efforts for data management middleware are aimed toward the "API between the client application and the middleware or the formats and protocols (FAPs) of the message between the middleware client and the middleware server." (Schulte, 1994)

Object oriented technology promises to allow developers to assemble more than half of their new applications from existing components in a fraction of the time previously required (Borkovsky, 1994). This dream will not happen without standards (Borkovsky, 1994). Most canonical solutions to distributed data management are already object oriented and if current trends continue, most of the next generation software will be object oriented as well (Booch, 1994).

VII. PROTOTYPE PROJECT

A. PURPOSE

The primary purpose of this prototype project was to test whether a middleware development tool permitted client/server applications to be developed quickly and easily. The other objective was to gain practical experience in rapid application development (RAD) and client/server application development. This prototype was done in conjunction with another project whose purpose was to test whether small development teams could successfully use RAD products in client/server application development.

B. DEVELOPMENT TEAM

There were four members on this development team: three Naval Postgraduate School (NPS) students and one NPS staff member. While each member had different levels of experience in program development, none had client/server systems or client/server application development experience. Team member backgrounds were:

- Fifteen years experience programming and systems analysis and design, expert experience in third generation languages (PLI and COBOL) and expert in 4GL using FOCUS, training in Paradox, knowledge of DBase, educated in FORTRAN, Assembly and RPG, and uses SAS extensively to verify and report data;

- Numerous programming courses including ADA, C++, Cobal, Fortran, and PL1 programming and strong practical experience in structured design;

- Programming courses in ADA and Fortran, database development using Paradox, and programming in CGI scripting language for applications on the WWW;

- Courses in ADA programming course and database development using Paradox and practical experience in database development in dBase III+ .

59

C. PROJECT DESCRIPTION

This project's goal was to develop an application which allows authorized personnel access to NPS faculty resume information from a central repository. Resume information is used to identify faculty members' expertise and research interests (Hutton, 1993). Resume information for approximately 150 tenure track faculty and selected staff members is currently maintained manually by the NPS Office of the Dean of Research. Information on non-tenure track instructors is not maintained. The person responsible for maintaining the information uses a typewriter to type resumes from information supplied by the faculty member. The finished documents are then photocopied and distributed to departments. The departments are required to update their binders by removing old resume pages and replacing them with the new resumes. There is a requirement for staff members to update their resumes annually but this is not always done. Approximately 30 resume changes are made each year and there are approximately 50 copies of the resume book throughout NPS (Hutton, 1993 and McLaughlin, 1994).

This project has been the focus of two previous theses and a Process Action Team (PAT). Led by Professor Paul Marto, Dean of Research, the PAT analyzed the current system and surveyed NPS faculty. Their conclusions cited the need to continue to maintain faculty information and photographs, stream-line the updating process, broaden the accessibility to the information, and have a system that is easy to use. Based on these recommendations, Hutton began a project to develop a multimedia NPS Faculty and Staff Resume Book EIS system and McLaughlin completed the development of a Windows 3.1 compatible multimedia application using Asymetrix's Multimedia ToolBook authoring software. Their work was completed in March 1994 but the application was not deployed. Hutton and McLaughlin's thesis described the background and requirements for this project in detail, therefore only summary information is provided in this thesis.

Our development team did not duplicate the analysis work that was previously completed because it was reasonably current and the Research department reported that

there were no changes to the requirements already outlined. The primary point of contact was helpful but not hopeful that this application would result in a useful system. This attitude is understandable since previous studies and developments have not changed the long established manual system.

1. Requirements

The project was coordinated by the Research department and was our primary source of requirements information. They provided the development team with data and report format requirements. The data requirements were developed by the PAT team and are similar to the information currently maintained manually. It was also decided that this application should address the two biggest drawbacks to the current resume collection and distribution system which are the labor intensive data collection and distribution process and the limited distribution.

Our research revealed multiple database tables on numerous servers that held various aspects of faculty information. The validity of some of this information was questionable and not all of the databases were fully functional. Some of these tables were developed and maintained by a user who utilized Borland's Paradox 3.5 for DOS. Other databases contained sensitive information, such as salary, and the data owners would not allow links into their data. For these reasons, we developed new databases.

Another important specification identified is the ability to search the database using keyword data. This will allow the Dean of Research to locate faculty for research projects as well as coordinate the work of those with similar research interests. This feature also allows students to locate potential thesis advisors by identifying those professors who have interests in the student's thesis subject area.

The Research department envisions the continued use of the resume binders that are distributed throughout the school. This means that our application must produce printed resumes in a format that is compatible with the current binder. The printed resume

must contain the faculty member's biographical information and photograph. This compatibility with the manual legacy system may appear to be discouraging the movement to an electronic system but the wariness of the user is understandable. The previous work in this area has not altered the current process and until a robust distributed application is developed and fully deployed, the binders will continue to be used.

Hutton's recommendation to include video capability was not considered feasible at this time but the ability to play a voice message was recognized as an optional requirement. This will allow staff members to record a short message for the benefit of the on-line user. Additional design specifications include:

- The application must be a Windows based application.

- The application must be scalable to a client-server application that will access remote databases.

- "Time stamps" should be attached to each resume so the user of the system can determine when the information was last updated.

- The application should be user-friendly. This is measured by whether the end-user can successfully navigate the application without the use of a user's manual.

- A user's manual will be developed explaining how to install the application and how to enter resume data. The user's manual will be written without any assumptions on the part of the user's level of computer experience.

2. Users

There are potentially many users of this application. The Research department serves as the database administrator but all faculty and staff are potential users. If this information is extended to the World Wide Web (WWW), the possibility of users outside the confines of NPS also exists. The intent is to make this application available to the following users:

- Other faculty members at NPS to help identify colleagues with similar research interests.

- Students at NPS to help locate thesis advisors with research interest that are relevant to their thesis work.

- Potential research sponsors.

- Researchers at other campuses with similar research to promote the enhancement of interdisciplinary work.

3. Application Benefits

This application provides few quantifiable benefits. The costs associated with the saved labor is minimal and it is difficult, if not impossible, to calculate an increase in research funding due to increased access to faculty resumes. The real benefit of this application is in the non-quantifiable benefits it can provide. These include:

- Broader access to faculty member's resume will provide greater data availability,

- Greater resume visibility may prompt faculty members to update their resumes more frequently leading to improved data timeliness and accuracy,

- Through increased data availability, potential research sponsors can get a better understanding of NPS's faculty and their research interests, and

- Maintaining all data on a single database will eliminate the problem of data redundancy. When this database is accessed by a user, he/she can be guaranteed to have access to the most current version of a resume being searched.

4. Constraints

There was a requirement for the faculty database to be fully operational by September 8, 1995. This gave the development team approximately eight weeks to develop an application. constraining the development of an application to only eight weeks is consistent with the promise of middleware tools. In addition, that kind of time frame is

an application developer's imperative if software id to constantly reflect changing user requirements and enable the user of rapidly changing technologies. For these reasons, it was embraced as a test of this middleware tool's learning curve factor and in turn, its usability.

The fact that none of the project members were experienced in Delphi, Object Pascal, or Windows programming constrains the functionality that can be attempted in such a short development time. This team decided to concentrate on the application's robustness and portability while implementing as many user requirements as possible. It was felt that these two qualities, robustness and portability, were most important if the application is to be expanded to a client/server architecture.

5. Software Used

The database schema was generated using Salsa which was then converted to Paradox for Windows, ver. 5.0, database tables. Salsa was selected to model the data because of its object oriented nature, its usability, and its ability to transform the model into Paradox tables. Salsa also provided all the functionality needed for this simple project, therefore, we were able to stay with a modeling tool that was familiar to most members rather than learn a new tool.

Paradox was chosen as the database development tool for a number of reasons. Although it is not considered a high powered database management system, it provides all the functionality this project required. Paradox does work in a multi-user environment and supports password protection for the data. Paradox is capable of accommodating simultaneous 20-25 users in a network environment. Since this prototype is not about database development, that part of the prototype development was considered incidental. This is not to de-emphasize the importance of the database in a client/server application, but to focus on the use of the middleware tool. With this in mind, Paradox was chosen

because the development team has developed databases using Paradox previously and could develop the databases quickly. The majority of the effort was to develop the application that would access the tables.

User interfaces were developed using Borland's Delphi for Windows, ver. 1.0. Delphi is a RAD and database development tool for Microsoft Windows. Delphi's language, Object Pascal, is based on the original Pascal language developed more than ten years ago. It includes a compiler that produces compiled code rather than interpreted code. Delphi includes a language compiler Delphi was chosen because it was the most powerful tool within an economic constraint. It also has a lower learning curve than other RAD tools with similar functionality. Its low cost, relatively low learning curve, and wide availability made it the tool of choice for this prototype.

D. EVALUATION OF DELPHI AS A MIDDLEWARE TOOL

Delphi enables application developers to visually create applications using components. It also permits a developer to build custom components within the same environment and use the same language. The Delphi Component Writer's Guide describes components as the "building block of Delphi applications." They are similar to C++ class libraries and are defined on three levels: functional, technical, and practical. The functional aspect is how the component is used from an end-user perspective. In this case, the end user is another programmer. The technical level is its inheritance from the Delphi supertype TComponent and the additional functionality the component developer adds. This inheritance provides a framework within which all components must operate. From a practical level, a component is any element that provides some type of functionality when inserted into an application. This functionality can be simple or complex, visual or non-visual. Component writing is not for the casual Delphi user or the programming novice. Because new components are descendants of other objects, component development requires a high level of understanding of objects and, more specifically, Delphi objects.

For the past ten weeks, I have spent many hours reading Delphi specific news-groups, WWW sites, and list groups. A sample of these Delphi resources are listed in Appendix A. I found these to be extremely helpful in learning to use Delphi. Much of the available information was not needed for this project but it gave me insight into Delphi's versatility, scalability, and extensibility. The newsgroups also provided me with a broader view of the advantages and disadvantages of using this product. The following two sections describe the views I found most commonly expressed.

1. Advantages

While there is a learning-curve associated with Delphi, it has been rated a success by many developers. The amount of time it takes to become comfortable with this new development environment depends on a programmer's background. It enables them to offer products and services that could not be offered before and to do so quickly enough to be very economical. The Delphi implementation is solid and logically designed. Because it's language is based on Turbo Pascal, which Borland has worked with for over a decade, it is a very reliable product.

An application developed in Delphi is faster than an application developed with an interpreted language such as Visual Basic. The time you spend in database DLLs is not affected by Delphi therefore, if it is a database application, the difference in speed from an interpreted language is minimal due to the interaction with the database engine.

Delphi allows you to create your own reusable components which are compiled into an application's executable file. This promotes code reuse while not adding to the number of files that must accompany the application. Delphi supports the use of Visual Basic components but the VBX components reside outside of the executable file and must accompany your application. Several developers stated that their initial coding is more procedurally oriented but when they review their code, the patterns are clearer and they can develop components for those functions/procedures that are reused throughout their

application. This is also true for those who develop components because they require the identical or similar functionality for more than one project.

Since components are extensible, minor features/properties/methods can be published so another developer can create a new component using the original component's features. You can also use Visual Basic components and access C++ DLL files. This provides the developer with access to existing code and makes the transition from these two languages easier. As a programmer gains experience in Delphi and Windows programming, they can take advantage of Delphi's ability to hook into a Windows API at a low level and its support of Windows callbacks.

A critical advantage of Delphi over other development tools is its extensibility within its own environment. This means you can build components and install them into the environment. Delphi also has an open architecture that allows you to build experts, component editors, property editors, and hook in almost any tool to the Integrated Development Environment (IDE).

Once an application developer has become comfortable working in the Delphi environment, they find it quicker and easier to develop applications. Because Delphi can support calls to C++ DLLs, the C++ programmers are able to use their existing routines that access low-level system functions more elegantly than Object Pascal.

Delphi has the capability to use ASCII files to a limited degree as tables. The ASCII driver has the capability to translate the data values in an ASCII fixed-length field or a comma-delimited file into fields and values that can be displayed through a TTable component. The use of fixed-length ASCII data is relatively simple and straight forward. The use of variable-length comma-delimited files however, is much more difficult.

Delphi's popularity provides a new or experienced Delphi user with a wealth of information that is available through the Internet. There are numerous WWW sites that have freeware and shareware components as well as commercial demonstration products. There are several sites that provide technical information and tips that are well written. The Delphi newsgroups are another excellent source of information. The response is

usually quick and accurate but should not be relied upon as the sole source of assistance. It is a good idea to read all of the posts for several weeks so you can gain an understanding of what causes programmers the most difficulty and who provides the most reliable information. I found that I could use the concepts I gained from reading the newsgroups in solving unrelated programming problems.

2. Disadvantages

If you have never used Pascal before, there is a significant learning curve. The documentation provided with Delphi is minimal so programmers must use other sources to work through all but the most basic programming problems. There is no shortage of third-party instruction books, however. There is a lot of activity surrounding Delphi and the publishing market has been quick to respond. The experience level each book targets is specific enough to allow for the person who is new to programming, new to Delphi, or the experienced user. During this project I found several of these books to be quite useful but did not find any single book that answered all of my questions. Quite frustrating was the number of code errors or omissions in several of these books. Most of the errors I encountered were in the books targeted for new programmers. I found it better to rely on the manuals and on-line help for basic information. Even though I am not an experienced programmer, the advanced Delphi books provided excellent information and were well written. I was able to take concepts and code segments and apply them to a much simpler situation.

At this time, the third party market for Delphi components is small. This is quickly changing however. Because any programmer can create components, the inventory of freeware, shareware, and commercial components is growing rapidly. The quality of these components varies therefore it is a 'caveat emptor' market. In the case of freeware and shareware, many of the components come with original source code. Some commercial products also allow you to purchase the source code but this is usually at an additional

68

source code. This gives the programmer the ability to modify the component, correct code errors, or learn from the component's code.

The biggest complaint is with Delphi's report generator, Report Smith. Although it is powerful in some respects, it is difficult to work with, slow to run, and adds a tremendous overhead. To deploy an application that uses Report Smith, Borland requires the installation of a run-time version of Report Smith to be installed. It requires over 13 megabytes of disk space for full installation. For use in an application that requires only simple one page reports, this is offensive. One third party report component that has wide-spread use is Crystal Reports. It was developed originally for use with Visual Basic but there are several good Visual Component Library (VCL) components that make interfacing to it from a Delphi application easy. This is a commercial product but once purchased, it does not require a licensing fee for applications that use it.

Delphi does not support multiple inheritance. A programmer can get around this by deriving an object from the class which gives the bulk of the desired functionality and adding another class in the descendant's fields to add additional behavior. While not always the most elegant solution, it can be done to simulate multiple inheritance.

E. PROTOTYPE

A hybrid development methodology was used to develop this product. A structured methodology was used during the system analysis and database design phase. A RAD approach was used in the application design and implementation phase.

1. Data Model

The database structures used for this prototype are depicted in the database schema provided in Appendix A. There is a 1-M-M relationship between the staff, keywords, and publications data. A staff member may have more than one keyword and more than one publication but need not have either.

The databases were designed using SALSA, a semantic object database design tool. The purpose of the database is to hold information about faculty at NPS. The title "staff" is used in the database in place of the title "faculty". This was done to reduce the number of characters needed in the database relation names to support the Windows/DOS convention of file names being restricted to eight characters.

Faculty at NPS consist of civilian faculty members and military faculty members. Attributes about both types of faculty members are maintained in one table. The use of a "supertype-subtype" design with "Staff" being used as a supertype and "MilStaff" and CivStaff" being used as the subtypes was considered. This design was rejected because the subtype CivStaff had no unique fields and the subtype MilStaff would have only two unique fields. Instead, a single table is used to represent all staff members. While this design approach can lead to problems of "data sparsity" in the relation, it is considered a minimal problem because only two attributes are effected.

The relation Staff uses "StaffID" as the primary key. This complies with the convention used by another faculty database and will facilitate data sharing if they are linked. This is also the convention the primary user is familiar with and prefers. The relation also has the attribute "SSN". This field is built into the relation, but currently not used. It is intended to be an alternate key if another database is selected as the database server. In the relation the attributes "PhotoFile" and "VoiceFile" contain pointers to graphic and audio files.

The final two relations, Publications and Keywords, are required to represent "one-to-many" relationships. The Publications relation as depicted in Appendix A consists of three attributes; "StaffID", "Publication", and "Display". Display is a boolean field that allows the faculty member to determine if the publication is displayed on the printed resume. This is needed because the database will contain all of the faculty member's publications but will print only five publications. The relation Keywords is required to store searchable keywords associated with a faculty member. A keyword represents a faculty members research or teaching interests. As an exam-

A keyword represents a faculty members research or teaching interests. As an example, an instructor in the Computer Science Department might list "C++," "Object Oriented Design," and "Visual Graphics" as his/her keywords.

2. Implementation

Implementation was done using a rapid prototyping methodology. The first prototype shown to the user was a simple set of user interfaces with no built-in functionality. This was beneficial in that the user was able to comment on the look of the product and provide immediate feed-back that could be quickly incorporated into the product.

The second prototype shown to the user is a stand alone version of the system with limited functionality. This version supports the design specifications as outlined in Section C.1., but security and client-server features have not been built in. To support a client-server version of the application, security features must be built in that support restricted write access at the record level. Write access should only be given to the owner of the resume, an authorized departmental representative and the Dean of Research's office. This will safeguard against malicious alteration of resume information.

The third prototype which will not be developed due to time constraints should include the above mentioned security features, plus client-server capabilities where the application accesses databases that are located on a campus database server. Password security is the only modification required to transfer the databases to a server. The application already has the capability of locating the data wherever it is located but will need to have the database engine configured for networked databases. A read-only version of this application would be a appropriate extension for those NPS users that will never have write access. Once this is achieved, the databases should be connected

to the World Wide Web (WWW) to provide for the maximum distribution of the resume book. In summary, there are three enhancement recommendations:

- Use a third-party component for resume printing.
- Develop a client/server version and place the databases on a campus server.
- Develop a WWW access application.

The application has been installed at the Office of the Dean of Research. The primary user was walked through the application's features and shortcomings. The user will evaluate the program for acceptance and additional functionality requirements.

F. LESSON'S LEARNED

An inexperienced programmer will find it difficult to develop a robust for the Windows' environment. The Windows' environment demands an understanding of its use of resources and API calls. For any program but the most basic, this requires the programmer to understand how his program is affecting the Windows' environment. Even for this simple prototype, API calls were used to print a single page report. The use of these calls are outside the realm of Delphi and therefore not explained beyond the fact that this is an API call. At the end of this project we began to get the error "Out of System Resources". The documentation concerning this error was nonexistent. From reading various Delphi manuals and third-party Delphi programming books, we could only guess that we were not freeing system resources after use. Where we might be allocating system resources or when to free them was not clear. This type of knowledge is essential if someone is going to write a robust Windows program.

Most database management systems provide a command to permanently remove deleted records and recover the disk space. It would seem that a simple Delphi call to the database's built-in functionality would accomplish this but that is not what we found.. Appendix B is the code required to permanently remove deleted database records. This is

one example of where we could find only a low-level brute force method to perform a seemingly simple task. This code also illustrates how complex it can be to make a robust application. Extensive exception handling is essential if the application is to shield the user from the myriad of events that can go wrong. If an application is not robust, the user may not utilize the program. Printing single page reports also falls into this category of surprisingly difficult.

The on-line help was more useful than the printed manuals but trying to find specific information was difficult. A programmer must wind their way through a series of topics to find the little tidbit of information that might prove useful. A programmer must also be able to grasp principles and be able to apply them to more than one situation. Frequently, we would look for information on how to implement something dynamically at run-time only to have all the examples show static implementation or elude to the fact that you can do something at run-time but not provide an example. This was especially frustrating when trying to figure out how to pass run-time variable information to Delphi's Report Smith. In situations such as this, we either revised our original implementation plan or implemented an inelegant solution. We changed our implementation plans when we could not get the run-time version of Report Smith to work on the primary user's computer. Our solution was to remove the capability to print the results of a keyword search. One of our inelegant solutions was highlighting the active edit box. Instead of writing code that could apply to all boxes, we wrote code for each edit box on the screen. One block of code to highlight the box, another block of code to return it to its normal color. This was the major cause of our program's excessive use of system resources and slow execution and had to be corrected before delivery to the end-user.

Not all information is in the manuals or the on-line help but in additional text files which come with Delphi. The most commonly missed information is the application distribution information. One of the more common questions in the newsgroups is:

> "Which files/directories/drivers do I need to include on my installation disks
> to make the application work on a PC with only Windows installed ? Can I
> ship the files royalty free ?"

This information is not in the manuals but in the file DEPLOY.TXT, found in the Delphi application directory. This file describes what is involved in distributing Delphi applications that include database access using the Borland Database Engine (BDE), the default database engine that comes with Delphi.

Delphi also carries forward some principles of Borland Pascal (BP). These principles are not explained and many times not even discussed. While reading the various newsgroups and technical information files available, there were many references to BP 7.0 code and how it was the same in Object Pascal (OP). This leads me to the conclusion that previous experience in Pascal programming would make the transition to OP and Delphi smoother.

VIII. CONCLUSION/RECOMMENDATIONS

When designing a client/server architecture, commands must look at the system as a whole and try to envision the possible impact that changes can have to the system. The architectural design a command chooses will impact such things as security and bandwidth requirements. Commands should begin their move toward client/server computing through small departmental projects so they can gain the experience needed to address an enterprise-wide venture. Commands should also realize that client/server computing is a rapidly changing environment. As the technology matures and standards stabilize, future incompatibility problems may arise. Commonly described as the glue between the client and the server, middleware tools can resolve many of these compatibility issues.

Traditionally, software and information systems have been developed using the system development life cycle (SDLC). This methodology consists of requirement specifications, system analysis and design, coding, testing and implementation. Depending upon the SDLC method, these are either distinct phases (Waterfall) or iterative phases (Spiral). Added to these methodologies is the use of RAD and prototyping. The use of RAD products enable organizations to rapidly adjust to changing business processes. Organizations can integrate the user in the development process to produce a product that meets the user's needs. Applications can be modified quickly to respond to a change in user requirements.

Mace (1994) stated that GUI point and click middleware is not as flexible as traditional programming middleware applications. New RAD middleware tools, such as Delphi, are providing more flexibility and control for the developer. If you go beyond the visual programming environment, Delphi provides low-level system manipulation capability. These new RAD products give commands the flexibility to use in-house talent to develop administrative applications rather than contracting for the same application. The use of in-house talent is beneficial if an organization takes a holistic approach to planning its information systems. With little to no training, talented users can develop

75

useful administrative applications. These applications can provide custom interfaces and reports for local databases. By determining which functions can reasonably be developed in-house and insisting on complete development and user documentation, commands can keep the IT structure from becoming a maintenance nightmare. As with client/server systems, commands should start with small, simple applications.

I would not rely on standards achieving the plug-and-play environment envisioned by some of the market optimists. SQL has been in the commercial market since Oracle's database in 1979. The first SQL standard was issued in 1986. After almost ten years SQL products from different vendors cannot communicate without vendor specific middleware or custom programming. Can we expect OMA compliant products to be any better? The first products to use ORBs do not communicate with each other. Standards are evolving and products are getting better, but plug-and-play compatibility will not materialize soon.

Another compatibility concern is that Microsoft is not a member of the OMG consortium. Since Microsoft controls the desktop market and is moving into the networking market, compatibility with Microsoft products should always be in the product selection equation. The best answer may be that it is not needed, but it should always be considered.

Middleware contains tools that can bring great benefits to those who use them wisely and great misery to those who don't. While these products can provide heterogeneous inter-connectivity for a wide number of systems, they are not always the ideal solution. They are tools and, as with all tools, they must be selected to match the situation. Middleware adds a layer of complexity, a potential bottleneck point, and another piece of proprietary software in an IT system. Selecting a middleware tool should be done only after you have a clear map of your information system and your information goals. Patching two organizations together, with or without middleware, is not an inter-organization network. Through the re-engineering process, organizations gain an understanding of their organization, its processes, how IT can facilitate their processes, and if middleware should be a part of their IT structure.

I do believe however, that middleware is an ideal tool for and a key element in the migration of legacy systems to an open systems client/server architecture. The purpose of middleware is to eliminate the need for IS professionals to reinvent solutions each time they need to solve an integration problem. If deliberate choices are made, the use of middleware tools can provide the system flexibility and connectivity needed when migrating to a an open systems client/server architecture.

APPENDIX A: INTERNET RESOURCES

Newsgroups

    comp.lang.pascal.delphi.databases
    comp.lang.pascal.delphi.misc
    comp.lang.pascal.delphi.components

List Server

    Majordomo@brickell.bridge.net

    Delphi on the Fly - Delphi Mailing List Archive
    http://www.qns.com/html/delphi/

WWW Locations

    Aerosoft
    http://www.widewest.com.au/aerosoft/

    Bill White's Delphi Components
    http://www.destek.net/cybermkt.blwhite.htm

    Borland
    http://www.borland.com

    Borland Delphi Technical Support
    http://www.borland.com/TechInfo/delphi/index.html

    City Zoo
    http://www.mindspring.com/~cityzoo/cityzoo.html

    Delphi Component: TButtonBar
    http://www.moai.com/b_bar.html

    Delphi Hackers' Corner
    http://www.it.kth.se/~ao/DHC/

    Delphi Northbay SIG
    http://super.sonic.net:80/delphisig/index.html

Delphi: RADical Application Development for Windows
http://Super.Sonic/Ann/Delphi/

Delphi SuperPage - Freeware
http://sunsite.icm.edu.pl/archiev/delphi/freeware.html

Grumpfish Delphi Support
http://www.teleport.com/~grump/delphi.htm

LionKnight Software
http://www.crl.com:80/~kanishka/gclient.html

NewYork Delphi User Group
http://www.iscinc.com/nydug.html

Startech: Delphi Internet Components
http://www.neosoft.com/~startech/delphi/delphi.htm

The Delphi Connection
http://www.pennant.com/delphi.html

The Delphi Station
http://www.teleport.com/~cwhite/wilddelphi.html

The Delphi Super Page
http://sunsite.icm.edu.pl/delphi/

APPENDIX B: DATABASE SCHEMA

Staff Data Table

Field	Type	Length
StaffID * Key Field	A	10
Awards	M	180*
Education	M	180*
Email_Address	A	35
Experience(Other)	M	180*
Experince (NPS)	M	180*
Name_FName	A	14
Name_LName	A	15
Name_Mid_Int	A	1
Office Code	A	5
Phone_AreaCode	A	5
Phone_FaxNo	A	8
Phone_LocalNum	A	8
Rank	A	4
RsrchInt	M	180*
Service	A	4
SSN	A	11
Status	A	20
TeachInt	M	180*
Time_Stamp	D	

* Length of a Paradox Memo field is limited only to disk space

81

Keywords Data Table

Field	Type	Length
StaffID *Key Field	A	10
Keywords *Key Field	A	40

Publications Data Table

Field	Type	Length
StaffID * Key Field	A	10
Publication * Key Field	A	150
Display	B	

APPENDIX C: SAMPLE CODE

```
{Pack Database File}
unit TablPack;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, DB, DBTables, DbiTypes, DbiProcs, DbiErrs;

type
  { Exceptions for TablePack }
  EPackError = class(Exception);
  EPack_InvalidParam = class(EPackError);
  EPack_InvalidHndl = class(EPackError);
  EPack_InvalidDBSpec = class(EPackError);
  EPack_NoConfigFile = class(EPackError);
  EPack_NoSuchTable = class(EPackError);
  EPack_NotSupported = class(EPackError);
  EPack_UnknownTblType = class(EPackError);
  EPack_UnknownDB = class(EPackError);
  EPack_NeedExclAccess = class(EPackError);
  EPack_IncorrectTblType = class(EPackError);
  EPack_DBLimit = class(EPackError);

  TTablePack = class(TTable)
  private
    { Private declarations }
    function GetTableType: PChar;
    procedure PackDBaseTable;
    procedure PackParadoxTable;
  protected
    { Protected declarations }
    function Chk(rslt: DbiResult): DbiResult; virtual;
  public
    { Public declarations }
    procedure Pack;
  published
    { Published declarations }
  end;
```

83

```
procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Data Access', [TTablePack]);
end;

function TTablePack.GetTableType: PChar;
var
  { FCurProp Holds information about the structure of the table }
  FCurProp: CurProps;

begin
  { Find out what type of table is currently opened.  NOTE: This is
    different than TTablePack.TableType }
  Chk(DbiGetCursorProps(Handle, FCurProp));
  GetTableType := FCurProp.szTableType;
end;

procedure TTablePack.Pack;
var
  TType: array[0..40] of char;

begin
  { The table must be opened to get directory information about the table
    before closing }
  if Active <> True then
    raise EPackError.Create('Table must be opened to be packed');
  { Get the type of table }
  strcopy(TType, GetTableType);
  if strcomp(TType, szParadox) = 0 then
    { Call PackParadoxTable procedure if PARADOX table }
    PackParadoxTable
  else
    if strcomp(TType, szDBase) = 0 then
      { Call PackDBaseTable procedure if dBase table }
      PackDBaseTable
    else
```

84

```
    { PARADOX and dBase table are the only types that can be packed }
    raise EPack_IncorrectTblType.Create('Incorrect table type: ' +
                            StrPas(TType));
end;

procedure TTablePack.PackParadoxTable;
var
  { Specific information about the table structure, indexes, etc. }
  TblDesc: CRTblDesc;
  { Uses as a handle to the database }
  hDb: hDbiDb;
  { Path to the currently opened table }
  TablePath: array[0..dbiMaxPathLen] of char;

begin
  { Initialize the table descriptor }
  FillChar(TblDesc, SizeOf(CRTblDesc), 0);
  with TblDesc do
  begin
    { Place the table name in descriptor }
    StrPCopy(szTblName, TableName);
    { Place the table type in descriptor }
    StrCopy(szTblType, GetTableType);
    { Set the packing option to true }
    bPack := True;
  end;
  { Initialize the DB handle }
  hDb := nil;
  { Get the current table's directory.  This is why the table MUST be△
    opened until now }
  Chk(DbiGetDirectory(DBHandle, True, TablePath));
  { Close the table }
  Close;
  { NOW: since the DbiDoRestructure call needs a valid DB handle BUT the
    table cannot be opened, call DbiOpenDatabase to get a valid handle.△
    Just leaving Active = FALSE does not give you a valid handle }
  Chk(DbiOpenDatabase(nil, 'STANDARD', dbiReadWrite, dbiOpenExcl, nil,
                0, nil, nil, hDb));
  { Set the table's directory to the old directory }
  Chk(DbiSetDirectory(hDb, TablePath));
  { Pack the PARADOX table }
```

```
  Chk(DbiDoRestructure(hDb, 1, @TblDesc, nil, nil, nil, FALSE));
  { Close the temporary database handle }
  Chk(DbiCloseDatabase(hDb));
  { Re-Open the table }
  Open;
end;

procedure TTablePack.PackDBaseTable;
begin
  { Pack the dBase Table }
  Chk(DbiPackTable(DBHandle, Handle, nil, nil, True));
end;

function TTablePack.Chk(rslt: DbiResult): DbiResult;
var
  ErrInfo: DbiErrInfo;
  ErrStr: string;
  pErr: array[0..dbiMaxMsgLen] of char;

begin
  { Only enter the error routine if an error has occured }
  if rslt <> dbiErr_None then
  begin
    { Get information on the error that occured.  ALWAYS DO THIS FIRST!! }
    DbiGetErrorInfo(False, ErrInfo);
    if ErrInfo.iError = rslt then
    begin
      ErrStr := Format('%s ', [ErrInfo.szErrCode]);
      if StrComp(ErrInfo.szContext[1], '') = 0 then
        ErrStr := Format('%s %s', [ErrStr, ErrInfo.szContext[1]]);
      if StrComp(ErrInfo.szContext[2], '') = 0 then
        ErrStr := Format('%s %s', [ErrStr, ErrInfo.szContext[2]]);
      if StrComp(ErrInfo.szContext[3], '') = 0 then
        ErrStr := Format('%s %s', [ErrStr, ErrInfo.szContext[3]]);
      if StrComp(ErrInfo.szContext[4], '') = 0 then
        ErrStr := Format('%s %s', [ErrStr, ErrInfo.szContext[4]]);
    end
    else
    begin
      DbiGetErrorString(rslt, pErr);
      ErrStr := StrPas(pErr);
    end;
```

```
ErrStr := Format('Table Pack Error: %d.  %s', [rslt, ErrStr]);

MessageBeep(mb_IconExclamation);
{ Raise the corrisponding exception }
case rslt of
  dbiErr_InvalidParam:
    raise EPack_InvalidParam.Create(ErrStr);
  dbiErr_InvalidHndl:
    raise EPack_InvalidHndl.Create(ErrStr);
  dbiErr_InvalidDBSpec:
    raise EPack_InvalidDBSpec.Create(ErrStr);
  dbiErr_NoSuchTable:
    raise EPack_NoSuchTable.Create(ErrStr);
  dbiErr_NoConfigFile:
    raise EPack_NoConfigFile.Create(ErrStr);
  dbiErr_NotSupported:
    raise EPack_NotSupported.Create(ErrStr);
  dbiErr_UnknownTblType:
    raise EPack_UnknownTblType.Create(ErrStr);
  dbiErr_UnknownDB:
    raise EPack_UnknownDB.Create(ErrStr);
  dbiErr_NeedExclAccess:
    raise EPack_NeedExclAccess.Create(ErrStr);
  dbiErr_DBLimit:
    raise EPack_DBLimit.Create(ErrStr);
  else
    raise EPackError.Create(ErrStr);
  end;
 end;
end;
end.
```

LIST OF REFERENCES

Adler, R., "Distributed Coordination Models for Client/Server Computing", *Computer*, vol. 28, no. 4, 1994.

Aiken, P., Muntz, A., and Richards, R., "DOD Legacy Systems Reverse Engineering Requirements", *ACM*, 1994.

Barbagallo, T., "Intelligent Middleware", *AI Expert*, vol. 9, no. 9, 1994.

Birkhead, E., "Middleware Unplugged", *INTERNETWORK*, vol. 5, no. 9, 1994.

Booch, G., "Coming of Age in an Object-Oriented World", IEEE Software, vol. 11, no. 6, 1994.

Borkovsky, E., "Battle of the Objects", *Computing Canada*, Sept 1, 1994.

Clancy, G., "A Comparative Study of Commercial and Department of Defense strategies for Developing Software Applications", *Naval Postgraduate School*, Sept 1994.

Classe, A., "Cleverwares", *Computer Weekly*, Aug 4, 1994.

Colonna-Romano, J., and Srite, P., *The Middleware Source Book*, Digital Press, 1995.

Douglas, L. and Ghoshal, K., "Integrating Third-Party Packages into an EMS", *IEEE Computer Applications in Power*, vol. 8, no. 2, 1995.

Endoso, J., "DOD Releases Long-Awaited Vision Plan for CIM Initiative", Government Computer News, Jan 1994.

Frank, M., "Cross-Platform Development", *DBMS*, vol. 7, no. 9, 1994.

Hart, P. and Estrin, D., "Inter-Organization Networks, Computer Integration", ACM Transactions on Information Systems, vol. 9, no. 4, 1991.

Hutton, P., "A Prototype of a Faculty and Staff Executive Information System", NPS Thesis, March 1993.

Kind, P. A. LTGEN, USA, "Software as a Force Multiplier", *Crosstalk*, vol. 7 no. 7, 1994.

Lewis, T., "Where is Client/Server Software Headed?", *Computer*, vol. 38, no. 4, 1995.

Lindholm, E., "Data Access without Chaos", *Datamation*, vol. 40, no. 15, 1994

Mace, S., "Windows Tool Links PC, Mainframe Applications", *InfoWorld*, vol. 16, no. 35, 1994

McLaughlin, Jr., R., "A Prototype of a Faculty and Staff Executive Information System", NPS Thesis, March, 1994.

Microsoft, "OLE Integration Technologies: Technical Overview", *Microsoft*, Oct 1994.

Orfali, R. and Harkey, D., "Client/Server Survival Guide With OS/2", *Van Nostrand Reinhold*, 1994.

Orfali, R., Harkey, D., and Edwards, J., "Essential Client/Server Survival Guide", *Van Nostrand Reinhold*, 1994.

OSF, "The OSF Distributed Computing Environment: Building on International Standards", *OSF*, 1992.

Palaniappan, M., et. al., "The Envoy Framework: An Open Architecture for Agents", *ACM Transactions on Information Systems*, vol. 10, no. 3, 1992.

Schulte, R., "Middleware: Panacea or Boondoggle?", *GartnerGroup*, July 5, 1994.

Stodder, D., "Slouching Toward Middleware", *Database Programming and Design*, vol. 7, no. 9, 1994.

Vowler, J., "Short Cuts Don't Pay", *Computer Weekly*, Sept 1, 1994.

Watterson, K., "Client/Server Technology for Managers", *Addison-Wesley*, 1994.

BIBLIOGRAPHY

Adler, R., "Distributed Coordination Models for Client/Server Computing", *Computer*, vol. 28, no. 4, 1994.

Aiken, P., Muntz, A., and Richards, R., "DOD Legacy Systems Reverse Engineering Requirements", *ACM*, 1994.

Barbagallo, T., "Intelligent Middleware", *AI Expert*, vol. 9, no. 9, 1994.

Birkhead, E., "Middleware Unplugged", *INTERNETWORK*, vol. 5, no. 9, 1994.

Booch, G., "Coming of Age in an Object-Oriented World", IEEE Software, vol. 11, no. 6, 1994.

Borkovsky, E., "Battle of the Objects", *Computing Canada*, Sept 1, 1994.

Clancy, G., "A Comparative Study of Commercial and Department of Defense strategies for Developing Software Applications", *Naval Postgraduate School*, Sept 1994.

Classe, A., "Cleverwares", *Computer Weekly*, Aug 4, 1994.

Colonna-Romano, J., and Srite, P., *The Middleware Source Book*, Digital Press, 1995.

Douglas, L. and Ghoshal, K., "Integrating Third-Party Packages into an EMS", *IEEE Computer Applications in Power*, vol. 8, no. 2, 1995.

Endoso, J., "DOD Releases Long-Awaited Vision Plan for CIM Initiative", Government Computer News, Jan 1994.

Frank, M., "Cross-Platform Development", *DBMS*, vol. 7, no. 9, 1994.

Hart, P. and Estrin, D., "Inter-Organization Networks, Computer Integration", ACM Transactions on Information Systems, vol. 9, no. 4, 1991.

Hutton, P., "A Prototype of a Faculty and Staff Executive Information System", NPS Thesis, March 1993.

Kind, P. A. LTGEN, USA, "Software as a Force Multiplier", *Crosstalk*, vol. 7 no. 7, 1994.

Lewis, T., "Where is Client/Server Software Headed?", *Computer*, vol. 38, no. 4, 1995.

Lindholm, E., "Data Access without Chaos", *Datamation*, vol. 40, no. 15, 1994

Mace, S., "Windows Tool Links PC, Mainframe Applications", *InfoWorld*, vol. 16, no. 35, 1994

McCarthy, S., "Why Agencies are so Slow to Embrace EDI", *Government Computer News*, vol. 14, no. 6, March 20, 1995.

McLaughlin, Jr., R., "A Prototype of a Faculty and Staff Executive Information System", NPS Thesis, March, 1994.

Microsoft, "OLE Integration Technologies: Technical Overview", *Microsoft*, Oct 1994.

Moffatt, C., "Designing Client-Server Applications for Enterprise Database Connectivity", *Microsoft*, 1994.

Montelone, F., "Middleware Casino", *Computerworld*, vol. 28, no. 36, 1994.

Moore, J., "Vendors Bolster Fed Efforts", *Federal Computer Week*, vol. 8, no. 26, 1994.

Orfali, R. and Harkey, D., "Client/Server Survival Guide With OS/2", *Van Nostrand Reinhold*, 1994.

Orfali, R., Harkey, D., and Edwards, J., "Essential Client/Server Survival Guide", *Van Nostrand Reinhold*, 1994.

OSF, "The OSF Distributed Computing Environment: Building on International Standards", *OSF*, 1992.

Palaniappan, M., et. al., "The Envoy Framework: An Open Architecture for Agents", *ACM Transactions on Information Systems*, vol. 10, no. 3, 1992.

Pfleeger, C., "Security in Computing", *Prentice Hall*, 1987.

Rabinovitch, E., "Mainframe Advocacy Sans Bag", *Datamation*, vol. 40, no. 16, 1994.

Schulte, R., "Middleware: Panacea or Boondoggle?", *GartnerGroup*, July 5, 1994.

Spitzer, T., "The Borland Potential", *DBMS*, vol. 7, no. 9, 1994.

Stodder, D., "Slouching Toward Middleware", *Database Programming and Design*, vol. 7, no. 9, 1994.

Vaughn, J., "Middleware: Up From Consultancies", *Software Magazine*, vol. 14, no. 9, 1994.

Vowler, J., "Short Cuts Don't Pay", *Computer Weekly*, Sept 1, 1994.

Watterson, K., "Client/Server Technology for Managers", *Addison-Wesley*, 1994.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center   2
   Cameron Station
   Alexandria, CA 22304-6145

2. Library, Code 052   2
   Naval Postgraduate School
   Monterey, CA 93943-5002

3. Professor Barry Frew Code SM/Fw   1
   Department of Systems Management
   Naval Postgraduate School
   Monterey, CA 93943

4. Professor Suresh Sridhar Code SM/Sr   1
   Department of Systems Management
   Naval Postgraduate School
   Monterey, CA 93943

5. LT Karen Weiss   2
   Naval Medical Information Management Center
   Building 27
   8901 Wisconsin Ave
   Bethesda, MD 20889-5605

6. Ms Charleen Johnson   1
   Defense Information Systems Agency
   Office of Technical Integration
   5201 Leesburg Pike, Suite 1501
   Falls Church, VA 22041-3201

7. Dr. M. Mestrovich   1
   Defense Information Systems Agency
   Office of Technical Integration
   5201 Leesburg Pike, Suite 1501
   Falls Church, VA 22041-3201